ISSN # 0748-9331

THE COMPUTER JOURNAL®

For Those Who Interface, Build, and Apply Micros

Issue Number 19 July—August, 1985 $2.50 US

Using the Extensibility of FORTH

Using the CREATE. . . . DOES Construct page 2

Extended CBIOS page 9

A $500 Superbrain Computer page 14

BASE:

Part Seven in a Series on

How to Design and Write Your Own Database page 17

Interfacing Tips and Troubles:

Part Two

Communicating with Telephone Tone Control page 19

Multitasking and Windows With CP/M-80

A Review of MTBASIC page 26

The Computer Corner page 32

THE COMPUTER JOURNAL
190 Sullivan Crossroad
Columbia Falls, Montana
59912
406-257-9119

Editor/Publisher
Art Carlson

Production Assistant
Judie Overbeek

Circulation
Donna Carlson

Technical Editor
Lance Rose

Contributing Editor
Ernie Brooner

Contributing Editor
Neil Bungard

Contributing Editor
Bill Kibler

The Computer Journal® is a bimonthly magazine for those who interface, build, and apply microcomputers.

The subscription rate is $14 for one year (6 issues), or $24 for two years (12 issues) in the U.S. Foreign rates on request.

Entire contents copyright © 1985 by The Computer Journal.

Advertising rates available upon request.

To indicate a change of address, please send your old label and new address.

Postmaster: Send address changes to: The Computer Journal, 190 Sullivan Crossroad, Columbis Falls, Montana, 59912.

Address all editorial, advertising and subscription inquiries to: The Computer Journal, 190 Sullivan Crossroad, Columbia Falls, MT 59912.

Editor's Page

We've Made Some Changes

You've undoubtedly noticed the delay between issues 18 and 19 (at least I hope that you missed us), and the reason for the delay is that we have changed the publishing schedule from monthly to bimonthly. We were never able to keep up with the monthly schedule and finally decided that it would be better to publish a larger, better, magazine every other month rather than slip a few days further behind every month. You will still receive the same number of issues that you paid for, but your subscription will last longer and reduce your annual cost.

We have also relocated to a larger, lower cost, out-of-town location where we can consolidate the publishing, electronics, and experimental machine shop. Before the move we were splitting our time and equipment between two locations, and what we needed always seemed to be at the other place. Once we're settled in we'll be more comfortable in the rural setting where I can see the snow capped mountains from the window (the base of the mountains is only 2½ miles away), but there are some distractions—such as when we have to stop to watch the bull elk lead his herd of 32 girls across the field. There is also a shortage of phone lines which means that we'll have to wait until another private line is available before we can proceed with the bulletin board.

The extra time between issues will allow us to produce a larger magazine with more time-consuming technical articles, and we'll be able to work more closely with our authors. We also intend to increase our contact with you, the reader, to find out what you are doing, and how we can better serve you. The only purpose of this magazine is to provide a place for everyone to share their knowledge with others, and we have not been getting enough feedback from YOU. We need to hear about your problems, your solutions, bugs you've discovered and how you patched them, bugs that you haven't figured out how to patch, and your tips, suggestions, and comments.

Everything doesn't have to be a major article, short notes and letters are also welcome. We had planned on setting up a bulletin board to make data exchange easier, but won't be able to do that until we can obtain a second private line. Would it help if we had a 300 Baud modem to put on line after answering the phone by voice?

Technical Software Exchange

There are a lot of user's groups providing public domain software, but we need a place to exchange the specialized programs that we need for engineering, scientific, interfacing, operating system modifications, robotics, measurement, control, numeric machining, and other odd-ball pursuits which interest us. Without a bulletin board there is the problem of disk formats, but I am going to start by offering to handle Apple II, CP/M-80 in 8" SSSD and some 5¼" formats, and probably IBM-PC disks. I am also going to try to have either the magazine or the author make disks of the program listing from the articles available to avoid the problems in rekeying long listings. If any of you would like to make a bulletin board available, it would be greatly appreciated by all of us.

"Knowledge is for sharing. Are you willing to contribute software or information for the exchange?"

It has also been suggested that we publish abstracts of articles which are too specialized to publish in the magazine, and provide photocopies or text files on disk of these articles to the interested parties for a small fee. This is another possible application for a bulletin board, but for those of us in remote rural areas the time charges for downloading long files is excessive and most articles will include illustrations which are difficult to handle over the modem.

(continued on page 30)

Using the Extensibility of FORTH

Using the CREATE. . . . DOES Construct

by Mahlon G. Kelly and Nicholas Spies

FORTH is a very powerful, fast, high-level language that is well suited for all sorts of general programming. And, for several reasons, it is better for interacting with hardware than anything but machine and assembler programming. One reason for this is that FORTH makes direct memory and port access very easy, but another is that a FORTH program is actually an extension of the language itself, able to do literally anything that the computer can do. Further, FORTH gives you the ability to add whole new features to the language, in particular, functions that are able to define new functions, analoguous to being able to create a function that can be used to create whole new types of subroutines in another language. This ability is unique to FORTH and provides power way beyond that given by any other language, yet it is an ability that is not well understood by those who could profit greatly by using FORTH. We will explore FORTH's extensibility in detail, but first we should describe some of the general characteristics of the language.

For several reasons FORTH is an unusual language. First, it uses a stack for all transfer of numbers between routines, including arithmetic and such things as transfer of ASCII characters and comparison operators. Second, it is very interactive, responding immediately to keyboard input. And third, as we have said, a FORTH program is a direct extension of the language. These unusual features provide FORTH with much of its power, but have given it a reputation for being hard to learn. FORTH is not really hard to learn—children take to it easily—but it does require that you not think of it like other languages; you must approach FORTH with an open mind. We can best show FORTH's unusual features with an example.

Suppose you want to calculate 25 squared + 33. If you typed

25 DUP * 33 + . (enter)

from the keyboard you would see 658 (the correct answer) displayed. What happened? Typing 25 placed 25 on the stack, **DUP** duplicated that number,leaving two 25s on the stack, one on top of the other. * then multiplied the numbers, leaving the product, 625, on the stack. 33 placed 33 on top of the 625, + added 33 to 625 giving 658, and . (pronounced "dot") finally caused the 658 to be displayed on the screen, emptying the stack. This can be diagrammed as follows:

Typed	STACK	
on keyboard	(top)	(bottom)
25	25	
DUP	25	25
*	625	
33	33	625
+	658	
.	(empty)	

This is a so-called last-in-first-out or LIFO stack. It produces what is known as *postfix* or *Reverse Polish Notation* (RPN), with the operators following the numbers, as opposed to the more common algebraic notation, with the operators between the numbers. Although RPN may initially seem difficult, with a little practice it becomes very natural. It has the advantage that parentheses are never needed to specify precedence, which is much simpler for the computer.

This example should make the interactive nature of FORTH clear; it can be used directly from the keyboard, even without writing a program. A more important point is shown by the operators **DUP**, *, +, and . . These are a few of many FORTH *words*. FORTH consists mainly of a dictionary of words resident in memory. When a word is input, the dictionary is searched until the word and its definition are found. The definition consists of a pointer or sequence of pointers to machine language routines that are executed to perform the function of the word, for example multiplying two numbers or displaying the top number on the stack. Since the machine language routines are fast, FORTH is fast. Virtually all of the functions of FORTH are performed by words, and writing a FORTH program simply consists of adding more words to the dictionary. Here's an example.

Suppose you must repeatedly square a number that is on the stack. You could define a new word **SQUARE** simply by inputting

: SQUARE DUP * ;

: is a FORTH word that has the function of taking all the following input until the word ; is encountered and using that input to define a new word, the name of which is the first sequence of letters encountered, in this case **SQUARE** (all words must be separated by spaces). All following words up to ; are then compiled in the dictionary as a sequence of pointers that comprise the definition. Now if you were to input

5 SQUARE .

you would see 25 displayed. Of course the same thing would have been accomplished by

5 DUP * .

and normally definitions are more complex than this example, but you can now see why FORTH is called an extensible language.

Here's another example. Suppose you must repeatedly perform the function

$$Y = AX^2 + B$$

where A and B are variables with numbers stored in them and X is a number on the top of the stack that is to be replaced by Y. First the variables must be created by

VARIABLE A

and

VARIABLE B

Variables are not used nearly as much in FORTH as in other languages because so much can be done with the stack, but variables are important for what we will cover shortly. A variable in FORTH is really a word that returns an address where a number may be stored. (The same is true of all languages, but the user is not aware of the address.) This would store 625 in the variable **A**.

```
625 A !
```

625 put a number on the stack and **A** put an address on top of 625. **!** (pronounced "store") is a FORTH word which stores the number that is second on the stack at the address on the top of the stack. To get the value of A back to the stack you can input

```
A @
```

where **@** (pronounced "fetch") is a word that fetches a number from memory to the stack, using the address previously on the stack, in this case the address returned by **A**.

Now we can define the function.

```
: AX2_ + _B SQUARE A @ * B @ + ;
```

Notice that the previously defined **SQUARE** was used; a program is built up by defining words using previous words. Now if you typed

```
35 A !  56 B !
```

followed by

```
2 AX2_ + _B .
```

. (dot) would display 196, the correct answer. **AX2_+_B** is actually a very short program, and it is executed by typing its name. It is very fast because all it does is follow a thread between a sequence of machine language routines (and this is why FORTH is called a threaded interpretive language or TIL).

There is another way to set up a variable, using the word **CREATE**.

```
5 CREATE A ,
```

would have the same effect as the previous definition of **A** with the addition that 5 would be stored in the variable as an initial value. **CREATE** creates a new definition in the dictionary, with the name being the next set of characters, in this case **A.** , (comma) reserves 2 bytes in the body of the word just created and stores 5 there (two bytes are used for most numbers in FORTH although more may be used if a range larger than – 32768 to 32767 is needed). A word defined using **CREATE** has the effect of returning the address of its own body to the stack, in this case where 5 was stored.

It is also possible to store several numbers in the body of a word, for example an array or a string of ASCII characters. The importance of **CREATE** for us here is that it can be used in conjunction with another word, **DOES>**, to define words that have the purpose of defining whole new sets of other words, greatly enhancing FORTH's extensibility, and providing an ability far beyond what is provided in any other language.

Parent And Child Words

The remarkable ability of FORTH to extend itself results from defining words. The sole purpose of defining words is to compile (define or create) other words. The most commonly used defining word is **:** (colon), and many short programs use nothing else. You have also seen **CREATE** and **VARIABLE**. When any defining word executes, it makes a new word by placing a header in the dictionary for the created word followed by whatever is needed for the new word to execute. The header contains the word's name and some other information. Every aspect of FORTH programming uses defining words to tie together more primitive routines to solve complex problems.

The important feature of FORTH that we will cover here is that you can create new defining words, that is, you are not limited to the defining words provided in the FORTH kernel. It's as easy to create new defining words as it is to create "regular" words. This opens almost unlimited possibilities for creating new types of words and new data types that can make programs more efficient and compact and that can also make programming easier.

Each defining word in the FORTH kernel is able to create a distinct class of words. For example, although each word defined with **:** (colon) performs a different function, they are all similar with respect to how they are defined and compiled and as to how they execute. All colon-words belong to a single class because they have all been created by **:** (colon) to combine the actions of more primitive words. Similarly all words created by **VARIABLE** are variables precisely because they all compile and execute in the same way. Thus every word may be classed according to what defining word was used to create it.

A way of keeping the relationship clear is to call defining words "parents"

FOR TRS-80 MODELS 1, 3 & 4
IBM PC, XT, AND COMPAQ

Train Your Computer to be an
EXPERT!

Expert systems facilitate the reduction of human expertise to simple, English-style rule-sets, then use them to diagnose problems. "Knowledge engineers" are developing many applications now.

EXPERT-2, Jack Park's outstanding introduction to expert systems, has been modified by MMS for MMSFORTH V2.0 and up. We supply it with full and well-documented source code to permit addition of advanced features, a good manual and sample rule-sets: stock market analysis, a digital fault analyzer, and the Animal Game. Plus the benefits of MMSFORTH's excellent full-screen editor, super-fast compiling, compact and high-speed run-time code, many built-in utilities and wide choice of other application programs.

(Rule 1 - demo in EXPERT-2)
IF you want EXPERT-2
ANDNOT you own MMSFORTH
THENHYP you need to buy
MMSFORTH plus EXPERT-2
BECAUSE MMSFORTH is required

EXPERT-2
In
MMSFORTH

The total software environment for IBM PC, TRS-80 Model 1, 3, 4 and close friends.

- Personal License (required):
  - MMSFORTH System Disk (IBM PC) $249.95
  - MMSFORTH System Disk (TRS-80 1, 3 or 4) 129.95
- Personal License (optional modules):
  - FORTHCOM communications module .... $ 39.95
  - UTILITIES 39.95
  - GAMES 39.95
  - EXPERT-2 expert system 69.95
  - DATAHANDLER 59.95
  - DATAHANDLER-PLUS (PC only, 128K req.) 89.95
  - FORTHWRITE word processor 175.00
- Corporate Site License Extensions ..... from $1,000
- Some recommended Forth books:
  - UNDERSTANDING FORTH (overview) .... $ 2.95
  - STARTING FORTH (programming) 19.95
  - THINKING FORTH (technique) 16.95
  - BEGINNING FORTH (re MMSFORTH) 16.95

Shipping/handling & tax extra. No returns on software.

Ask your dealer to show you the world of MMSFORTH, or request our free brochure.

MILLER MICROCOMPUTER SERVICES
61 Lake Shore Road, Natick, MA 01760
(617) 653-6136

and the words they create "children." Each child of a common parent-word has a behavior that is similar to, yet different from, that of its "siblings." The common behavior of siblings is a consequence of their sharing the same execution action as defined in their common parent. The differences of the children are a result of the different values compiled into them when they are created (for example different words, represented by different pointers, within a colon definition). To understand why a child-word acts the way it does you have to look both at its parent's definition and its own.

If the genealogy of words is charted, we can discern three stages, often called sequences in the FORTH literature. The sequences are what happens when: (1) The parent is born (compile defining word). (2) The parent acts and gives birth to the child (execute defining word and compile child-word). (3) The child acts (execute child-word).

The reason there are three stages instead of four is simple: whenever a defining word executes, it compiles a child-word. What seems like two stages is really one and the same action.

The general behavior of the children of a particular parent is predetermined by the definition of the parent word in stage 1. Colon words are all similar because of the way : itself was defined. They all execute in the same way because one of the actions of a defining word is to make its children all have similar behavior. This is most easily seen in the case of the children of **VARIABLE** and another word, **CONSTANT**. As you have seen, variables all put the address of their contents on the stack; constants on the other hand all put the contents themselves on the stack. For example if you defined

65 CONSTANT EXAMPLE

when **EXAMPLE** was executed 65 would be put on the stack; no **@** is used. The contents of a constant are set up when it is defined, and the contents are harder to change than those of a variable. They normally stay the same throughout the execution of a program. The difference in action is because of the way that **VARIABLE** and **CONSTANT** were each defined; we will give you their definitions shortly.

The differences in the behavior of siblings is thus determined during stage 2, when the defining word executes and compiles a child. Siblings do different things because they contain different things. Each colon word is distinct because it is defined using a different combination of words. Constants vary according to the values they were created with. So we can say that the children of a common defining word (siblings) are the same because they execute in the same way, but are different because they contain different values or addresses.

Thinking in terms of these three stages or sequences of a word's geneology will help to prevent confusion when new defining words are created.

To summarize:

> Sequence 1. A defining word is created to compile children with a certain type of behavior.
> Sequence 2. A defining word executes to create a child with specific contents and behavior.
> Sequence 3. A child-word executes according to what its parent told it to do with its contents.

It may seem rather mysterious that one word can determine how another word will execute, but it's really quite simple. When a defining word creates a child, in addition to storing the child's contents, it stores the address of the child's run-time code into the child. The run-time code is a machine language program that determines how the child will execute, that is, what it will do with its contents. Since each defining word stores the address of a specific run-time code in all of its children, all of its children execute in the same way.

Creating Defining Words

How are defining words defined? Some, like : and **CREATE** are part of the FORTH kernel, the elementary code that was produced when the language itself was produced with an assembler, cross-compiler or something of the sort. But how can the user create new defining words? The answer is in **CREATE**, either used by itself or with the word **DOES>**. **CREATE** by itself is the most fundamental way of defining new words in FORTH. **CREATE** is used as

CREATE CHILD-WORD

where **CHILD-WORD** is the word being defined. As mentioned above the action of **CREATE** is to place a header for **CHILD-WORD** in the dictionary, and when **CHILD-WORD** is executed, the address of its contents is put on the stack. **CREATE** itself does not allocate any dictionary space beyond the header of the child-word – this must be done as a separate step (with , (comma), for example, as used above). **VARIABLE** can be defined as:

: VARIABLE CREATE 0 , ;

When **VARIABLE** is executed **CREATE** acts, not to compile 0, which was compiled when **VARIABLE** was defined, but to compile the next word in the input stream. **VARIABLE** then puts 0 on the stack and , (comma) places the 0 as two bytes in the body of the new variable. Just as the actions of **DUP** and * in the definition of **SQUARE** were deferred until **SQUARE** was executed, the action of **CREATE** is deferred until **VARIABLE** is executed.

The stage 1 activity of **VARIABLE** is the definition. The stage 2 activity of **VARIABLE** is when it is used as in

VARIABLE DISCOUNT

which breaks down to

VARIABLE Start execution of defining word.
CREATE Make DISCOUNT a word in the dictionary. Store address of run-time code in **DISCOUNT**.
0 , Compile two zero-bytes into **DISCOUNT**.

The stage 3 activity of **VARIABLE** occurs when **DISCOUNT** executes. This is when the run-time code that was put into **DISCOUNT** by **CREATE** executes and puts the address of the contents of **DISCOUNT** onto the stack. *Any colon word that has* **CREATE** *as part of its definition is a new defining word.*

But how can **CONSTANT** be defined? It might at first seem that we could define **CONSTANT** as

: BAD-CONSTANT CREATE , @ ;

but immediately we can see that **BAD-CONSTANT** cannot work because the **@** will execute during stage 2, when the child is created, and not when the child-word is executed. What we actually want to do is define **CONSTANT** so that its children's contents are fetched in stage 3. That is done with **DOES>**.

Creating New Defining Words

The word **DOES>** is needed to permit defining words to determine the execution behavior of their children. Now we can define

: CONSTANT CREATE , DOES> @ ;

The stage 1 activity of **CONSTANT** occurs when this definition is compiled. When **CONSTANT** executes to compile another word, as with

```
1024 CONSTANT 1K
```

the stage 2 activity of **CONSTANT** can be mapped as

```
CONSTANT   Start execution of defining word.
CREATE     Make 1K a word in the dictionary. Store address of run-time code in 1K.
,          Compile the number 1024 from the stack.
```

But, because of the presence of **DOES>** **@** , we know that the stage 3 activity of **CONSTANT** (when **1K** executes) is more complicated than in the case of **VARIABLE** . When **1K** executes, the address of the contents of **1K** is first put on the stack (because **CREATE** put the run-time code in **1K** to do just that), and then **@** fetches the contents from this address, putting 1024 on the stack. In other words, the **@** following **DOES>** is executed when the child-word executes, not when it is defined. The function of **DOES>** is to specify that the words following it are to be executed when the child word executes, while those between **CREATE** and **DOES>** are executed when the child word is created.

The definition of **CONSTANT** is a good example of how new defining words are created. To reiterate, the words between **CREATE** and **DOES>** are executed during stage 2, when the parent executes and compiles the child-word. When the child-word itself executes, the address of its contents is first put on the stack. Then the words that followed **DOES>** in the defining word are executed, determining what the child-word "does."

Here's an application for defining words. Most terminals and printers are controlled with the ASCII values between 0 to 31 (called control codes). The control values could be stored in constants and output to the terminal with **EMIT** , a word that sends a character represented by a number on the stack to the terminal, but using a new defining word is more efficient. Consider:

```
: IS-CONTROL CREATE ,
    DOES> @ EMIT ;
```

IS-CONTROL is nothing but **CONSTANT** with **EMIT** added to its children's execution behavior. **IS-CONTROL** can be used to create a family of related words such as

```
 7 IS-CONTROL BELL
 8 IS-CONTROL BACKSPACE
12 IS-CONTROL FORMFEED
13 IS-CONTROL CR
```

where each word will tell a terminal to perform a certain action. That is, **BELL** would cause most terminals to sound a beep.

One advantage of defining words should already be apparent: They encourage readable programs. When **IS-CONTROL** is used, all that needs to be specified is the data to distinguish the new word from the other children of **IS-CONTROL** , specifically a control code and a name. Each child stands out as an individual in the family of **IS-CONTROL** definitions. Defining words encourages the separation of the behavior that related definitions have in common from their individual behavior. The common behavior of the children is coded in the **DOES>** portion of the defining word's definition. The individual behavior of each child is determined by the value (or values) on the stack when it is created. Problems that involve a series of words with similar definitions can often best be solved by creating a new defining word.

Here's another example. We earlier showed how to define a mathematical function with a colon definition. With a defining word you can create any number of linear equations of the form

```
y = ax + b
```

by creating children when the coefficients a and b are on the stack. When a child-word then executes with x on the stack it will leave the solution y. The definition of the defining word is

```
: LINER A ( a b - ) CREATE SWAP , ,
  DOES> DUP >R @ * R> 2+ @ + ;
```

Notice that the values on the stack (a and b) are swapped when **LINEAR** is used so as to save some stack manipulation when the child executes. You should always try to have work done during a definition if it will save time during execution of a child. If the linear equation

```
y = 3x + 17
```

is defined with

```
3 17 LINEAR ALINE
```

then if

```
2 ALINE .
```

is executed 23 will be shown as the solution. The definition needs some explanation. First notice that more than one number can be placed from the stack into the body of the child-word by using , (comma) repeatedly. **>R** is a word that takes a number from the

NGS FORTH

A FAST FORTH, OPTIMIZED FOR THE IBM PERSONAL COMPUTER AND MS-DOS COMPATIBLES.

STANDARD FEATURES INCLUDE:

- 79 STANDARD
- DIRECT I/O ACCESS
- FULL ACCESS TO MS-DOS FILES AND FUNCTIONS
- ENVIRONMENT SAVE & LOAD
- MULTI-SEGMENTED FOR LARGE APPLICATIONS
- EXTENDED ADDRESSING
- MEMORY ALLOCATION CONFIGURABLE ON-LINE
- AUTO LOAD SCREEN BOOT
- LINE & SCREEN EDITORS
- DECOMPILER AND DEBUGGING AIDS
- 8088 ASSEMBLER
- GRAPHICS & SOUND
- NGS ENHANCEMENTS
- DETAILED MANUAL
- INEXPENSIVE UPGRADES
- NGS USER NEWSLETTER

A COMPLETE FORTH DEVELOPMENT SYSTEM.

PRICES START AT $70

NEW◆HP-150 & HP-110 VERSIONS AVAILABLE

NEXT GENERATION SYSTEMS
P.O.BOX 2987
SANTA CLARA, CA. 95055
(408) 241-5909

main stack and places it on a second stack (called the return stack). **R>** pops the number from the return stack and places it back on the main stack. This lets the return stack serve as a temporary storage area. **2+** is a FORTH word that simply adds 2 to the number on the stack; it performs the action of **2 +** but is defined as a single word in assembler for speed. The execution of **ALINE** may now be shown as in Figure 1.

The material in parentheses shows the stack contents after the execution of each word; it is a FORTH convention to use such stack charts, and anything within parentheses (with the opening parenthesis followed by a space) is a comment in FORTH; it is ignored by the language.

This shows a common technique used in complex defining words. Because the address of the first cell of **ALINE** will be needed to fetch two numbers this address is stored on the return stack. When the address is then taken from the return stack it must be incremented by 2 so as to point to the next cell in **ALINE** allowing the 17 compiled there to be fetched. (In FORTH, a cell is the two bytes needed to store a number.) Although there are words for manipulating the stack that could have been used in this example, variations of this technique can be used to store and recover any number of bytes or numbers in the children of defining words.

The best way to become familiar with the power of defining new defining words is with practice. Here are some exercises that should give you some more ideas. The answers are at the end of the article.

Exercises

1) Define a word called **MAKEDATE** that expects month, day, and year numbers on the stack and creates children that will display their date. Thus

12 7 41 MAKEDATE
PEARLHARBOR

would create **PEARLHARBOR**, which would display 12 07 41 when it executes.

2) Define a defining word called **COUNTER** that uses the number on the stack to initialize its children. When the children of **COUNTER** execute they will modify their own contents to be one greater each time they execute. Thus

0 COUNTER COUNTIT

```
DUP   ( -- 2 addr addr )    Addr of 3 in ALINE .
>R    ( -- 2 addr )         Put addr on return stack.
@     ( -- 2 3 )            Fetch 3 (a).
*     ( -- 6 )              6 = ax, the first term.
R>    ( -- 6 addr )         Addr of 3 in ALINE .
2+    ( -- 6 addr+2 )       Addr of 17 in ALINE .
@     ( -- 6 17 )           Fetch 17 (b), the second term.
+     ( -- 23 )             23 = y = ax + b, the solution.
```

Figure 1

would create **COUNTIT** which would modify its contents so as to be 1, 2, 3 and so on each time **COUNTIT** executes. If a child of **COUNTER** is used in a colon-definition it will keep track of the number of times the colon-word is executed.

3) Define a word **QUADRATIC** that works similarly to **LINEAR** but defines children which, given x, will solve equations of the form

$$y = ax2 + bx + c$$

4) The Michaelis-Menten equation has wide application in biology and biochemistry for determining the rates of enzymatic reactions. Its general form is

$$Q = Q_{max}S / (K_m + S)$$

where Q is the rate of the reaction, S is the substrate concentration, Q_{max} is the maximum rate of the reaction, and K_m is the half-saturation constant or the substrate concentration at which the reaction rate is one half of its maximum. Write a defining word to solve this equation given Q_{max} and K_m on the stack when the child is created and S on the stack when it executes.

Defining Arrays

An array is simply a sequential list of numbers. Each number is called an element, and each element is numbered; that is, the first number in the list is either the first or 0th element, depending on whether one chooses to number the elements from 0 or 1. Arrays are very useful in a wide variety of fields and can be stored in memory in consecutive pairs of bytes. Arrays are an excellent application for defining words because the word **ARRAY** can be defined to be used to create a child array of a certain size which, when it executes, will return the address of a particular element to the stack.

Here's a parent word to create arrays. One way of defining **ARRAY** would be

```
: ARRAY  CREATE 2 *  ALLOT DOES>
    SWAP 2 * + ;
```

When used as

30 ARRAY NOVEMBER

an array of 30 numbers would be created with elements numbered from 0 to 29. When the child-word is used it expects an element-number on the stack with which to calculate an address. That is,

2397 1 NOVEMBER !

would calculate the address of the third byte in **NOVEMBER** and store 2397 there.

1 NOVEMBER @ .

would then fetch 2397 and display it. The word **ALLOT** allocates a number of bytes specified on the stack into the body of a word for number storage. Since each number is two bytes long, the number of bytes that is **ALLOT**ed must be twice the number on the stack; that is, 60 bytes must be **ALLOT**ed in **NOVEMBER**. When the child word is executed the address it returns must be **SWAP**ped on the stack with the element that is already stored there. That element is then multiplied by 2 to get the offset from the start of the storage area to where the number is stored. That offset is then added to the start of the storage to get the address where the number is stored.

There are two ways of numbering the elements of arrays, either starting with element 0 (as shown above) or starting with element 1. If you want **1 NOVEMBER** to give you the address of the first (rather than the second) number in the **NOVEMBER** array then define **ARRAY** as

```
: ARRAY  CREATE 2 * ALLOT DOES>
    SWAP 1- 2 * + ;
```

Although starting with element 1 is slightly slower it may be easier to work with.

Notice that arrays created with these simple definitions will dutifully return an address outside of the reserved array space if they are given an element-number that is out of range. Storing values outside of the alloted area can cause disastrous errors. The proper solution would be to write a word to make sure the array index is

within legal bounds before the index is handed to the array word. But incorporating error-checking in the definitions of **ARRAY** will slow the performance of each child array-word whenever it is used, whether or not there might be an error.

If you are familiar with matrices you may want to try this exercise. Write a word similar to **ARRAY**, called **2ARRAY**, that will create a two-dimensional array. Thus if

5 8 2ARRAY ARR

is executed, an array called **ARR** should be created to hold a matrix with 5 rows and 8 columns. Typing

3 6 ARR

should return the address of the number stored in the 3rd row and 6th column, counting from element 1. The answer is give at the end of the article.

An Applied Example — Using FORTH For Gathering Data

One of the most important uses of FORTH is for interacting with devices outside of the computer. Our final example will show how the **CREATE ... DOES>** construct can be used to help a computer gather and analyze data in real time. This example is derived from an actual application used by one of the authors for his research. In fact, he learned FORTH because no other language could be used as easily to do what he wanted.

The problem is to take data from a variety of water-quality sensors in a lake, put it into the computer, and then manipulate the data to see its behavior. There may be up to 48 sensing devices including photocells, pH electrodes, instruments to measure the clarity of the water (turbidimeters), flow measuring devices, and dissolved-oxygen electrodes. The modules containing the sensors are suspended in a lake, and we want to be able to see what is being measured as well as what was measured in the previous 24 hours. Cables lead from the sensors to a microcomputer on the shore. Each device produces an output voltage between 0 and 1000 millivolts, proportional to what is being measured. The output is fed to an analog to digital converter to produce a number made available on the computer's input ports. For our example we will assume that a FORTH program has been written to place the millivolt input into an array called **PORT-DATA**, such that 1 **PORT-DATA @** will return the current millivolt value from port 1, and so on. The data-gathering program is not difficult, but it depends on the computer and dialect of FORTH being used.

Our problem is that we want to convert the raw-voltage input to actual values of pH, temperature, and so on before it is passed to a word that will summarize that data in tabular and graphic form and store it on disk. We have an equation for each sensor. Here we will consider only the temperature measurements, but the procedure for the other parameters is nearly identical. Each temperature sensor produces a voltage that is linearly proportional to the temperature. That is,

$$T = aV + b$$

where T is the value of temperature, V is the voltage, and a and b are calibration constants that are different for each sensor. We need a word for each temperature sensor which, when it is executed, will take a voltage from the stack, calculate the temperature (actually the temperature scaled by a factor of 1000), and store the result in an array called **RESULT**. Those words, for example **1TEMP**, **2TEMP** and so on, can be created by a defining word as in

port# a b VOLT-TO-TEMP nTEMP

where **nTEMP** is the child-word. Thus if the equation for temperature transducer number 8 on port 32 is

$$T = 26v + 1200$$

then the appropriate conversion word would be created by

32 26 1200 VOLT-TO-TEMP 8TEMP

The definition for **VOLT-TO-TEMP** would then be as shown in Figure 2.

where the initial values are compiled in the same order as they occur on the stack. When the child-word (8TEMP) executes, it gets the raw voltage data for the appropriate port (32) and converts it to a temperature using the words following **DOES>**. Finally the results are stored in the previously created array called **RESULT**. The word **ROT** rotates the top 3 numbers on the stack, that is, the third number is moved to the top, the top to the second, and the second to the third. **R@** is similar to **R>** except that it only copies the number on the return stack to the main stack without removing it. Understanding this example may take some study, but it would be impossible to perform this task so simply in any other language, and the function would be slower and take more memory as well.

There would be equivalent defining words for each type of sensor, that is, for oxygen, pH, and so on. They would differ in the words following **DOES>** because the form of the equations relating the voltages to the actual values for each sensor-type is different.

Although this is only part of a program (getting the voltages, and displaying and storing the results are not simple), you can see that the child-words can easily be modified to accommodate changes of sensors or of their calibration. By using defining words the calibration values used for each instrument are clearly shown by the parameters used to create each child-word. This is an example of how defining words not only make source code more readable, but also how they help to design programs that are more structured and compact.

In Conclusion

As we said at the beginning, FORTH

```
: VOLT-TO-TEMP    ( Name of defining word.)
  CREATE          ( Compile a header for child.)
                  ( Put address of run-time code in child.)
  ROT , SWAP , ,  ( Compile as: port#, a, b.)
  DOES>           ( Start defining execution of child.)
                  ( Put address of child's contents on stack.)
  >R              ( -- ) ( Put address on return stack.)
  R@ @            ( -- port# ) ( Get port number in child.)
  PORT-DATA @     ( -- data ) ( Get voltage from port n.)
  R@ 2+ @         ( -- V a ) ( Get a.)
  *               ( -- V*a ) ( Add to data to start from zero.)
  R@ 4 + @        ( -- V*a b ) ( Get b.)
  +               ( -- result ) ( Calculate temperature.)
  R> @            ( -- port# result ) ( Get port number in child.)
  RESULT !        ( -- ) ( Store result in nth array position.)
  ;               ( End definition.)
```

Figure 2

is an unusual language. We have only touched on one of its features, but one of the most powerful and subtle. FORTH programs are written by extending the language, which means defining new words using, appropriately, defining words. Just the ability to define new words makes FORTH more powerful than other languages. But beyond, that one can define new defining words — words that create whole new classes of words. The end result is that very specialized applications can be created in FORTH. The creation of words for calibrating temperature transducers is an example. But the defining words could be for creating new phoneme translators in a speech synthesizer, controlling arm motions of a robot, or sorting objects using pattern recognition. Any set of tasks with similar but slightly different actions will benefit from the use of specialized defining words. This makes FORTH particularly well suited to hardware control or interfacing. In fact FORTH can be used to create specialized vocabularies, or even specialized pseudo-languages to perform functions in dedicated applications. Since FORTH code can be easily put in ROM it is well suited to dedicated control hardware. It can substitute for machine language programs in such situations.

We have only touched on FORTH's power. It is possible, for example, to merge assembler language instructions into FORTH words if no pre-existing FORTH words will do the job (which is rare). And we have left out many elementary features of FORTH. We have hardly mentioned the power of the stack, how to work with strings or floating point arithmetic, or how to exchange programs and data with a disk (which is simple). And we have not covered FORTH's remarkable ability to access memory and work with bytes and bits in variable number bases. FORTH can do anything that any other language can and more, and usually faster and with less memory. FORTH programs may actually use less memory than equivalent programs written in assembler language. ■

This article is based on a chapter from the book **FORTH, A Text and Reference,** *an introduction to fundamental and advanced aspects of the language that will be published by Prentice-Hall.*

```
 0 ( Answers to exercises, block 1 of 3 )   : TASK ;
 1
 2 ( EXERCISE # 1 )
 3 : MAKEDATE ( n1 n2 n3 --  ) CREATE , , , DOES>
 4      ( Child-word action: )
 5   DUP @          ( Fetch year  )
 6   SWAP DUP 2+ @  ( Fetch month )
 7   SWAP 4 + @     ( Fetch day   )
 8   . . . ;        ( Print them  )
 9
10 ( EXERCISE # 2 )
11 : COUNTER ( n -- )  CREATE , DOES> 1 SWAP +! ;
12   0 COUNTER COUNTIT
13      ( To get the count do ' COUNTIT @ . )
14
15
```

```
 0 ( Answers to exercises, block 2 of 3 )   : TASK ;
 1
 2 ( EXERCISE # 3 )
 3 : QUADRATIC ( a b c -- )  CREATE , , , DOES>
 4      ( Child-word action, child stack chart ( x -- y )
 5   SWAP >R ( >R places x on another stack, the "return stack" )
 6   DUP @          ( addr c )
 7   SWAP DUP 2+ @ ( c addr b )
 8   SWAP 4 + @     ( c b a )
 9   R@ DUP * *     ( c b ax^2 ) ( R@ fetches x from return stack )
10   SWAP R> *      ( c ax^2 bx ) ( R> removes x from return stack )
11   + + ;          ( y )
12 ( Using 3 2 1 QUADRATIC QUAD , when 2 QUAD . is executed
13   17 should be displayed. )
14
15
```

```
 0 ( Answers to exercises, block 3 of xxx )   : TASK ;
 1
 2 ( EXERCISE # 4 )
 3 : MICH-MENT ( Qm Km -- ) CREATE , , DOES>
 4   SWAP >R DUP @ SWAP 2+ @    ( Km Qm )
 5   R@ * ( Km Qm*S )
 6   SWAP R> + / ; ( Q )
 7
 8 ( EXERCISE IN TEXT ON PAGE  6 )
 9 : 2ARRAY ( rows cols -- ) CREATE DUP , * 2 * ALLOT DOES>
10   >R SWAP R@ @ * + 1+ 2 * R> + ;
11   ( The trick is to store the number of columns as the first
12     element in the array, and then let the child word use it
13     to calculate the offset in the array. )
14
15
```


MODEL 100 C COMPILER

Now you can write efficient programs for your TRS-80 model 100 with ease. Or, learn the essentials of C programming while traveling!

C/100 - THE "PORTABLE" C COMPILER

Cassette version $49.00
Disk/Video interface version $59.00
Model II version (run on mod II, then download object code to model 100) .. $79.00
Model III version (as above for Mod III) .. $79.00

Write or call for information on other TRS-80 software

MODELS II, 12, 16
MODELS III, 4

TRS/C C COMPILER

Full K&R with source to the function library. UNIX compatible $85.00

ZSPF EDITOR

SPF, the choice of most mainframe programmers, is now available for Z80 machines. And it's panel driven so you can customize it! $75.00

business utility software
109 minna ste 423 san francisco ca 94105
(415) 397-2000

EXTENDED CBIOS

by Everson J. Ecoff

Background

This project began with a desire to add a very large CBIOS (Custom Basic Input/Output System) driver to a CP/M based system in order to make use of a Selectric typewriter as a list device. Because the Selectric has a minimal hardware interface, it requires a long (400 byte) software driver. It seemed undesirable to make room for the CBIOS driver by moving the CP/M system down, as this would greatly reduce the size of the user TPA (Transient Program Area). The approach that seemed most feasible was to form a two-piece CBIOS. One small module would reside in the place of the usual CBIOS and would be accessible to all users. The second, larger module would reside in some bankable area of memory and would only be switched in by the resident CBIOS when needed.

The first decision that had to be made concerned the type of bank switching. Close examination of the system revealed that the Digital Research Computers 64K static memory board allowed the S-100 bus phantom* line (pin 67) to disable the lower 32K of memory. This fact set the stage for the approach that was finally used. Figure 1 displays the relationship of the deselectable memory to the system memory map. It became obvious after some study that the idea was going to require some refinement in order to be useable. Figure 2 shows the next required refinement in the implementation. Notice that the area of memory that is allowed to be disabled is a portion of the user TPA. The memory locations below 100H that are assigned for system use will not be disabled under any circumstances. Only the user TPA from 100H to 8000H will be disabled by the exertion of the phantom* line. The final requirement was to provide a memory substitute that would take the place of the disabled portion of the TPA. Figure 3, which shows the maximum size of the alternate TPA, illustrates the desired final requirements. The actual implementation only needs to utilize an alternate TPA size that is sufficient to accomplish the desired enhancements. I decided to use a 12K byte size in my implementation.

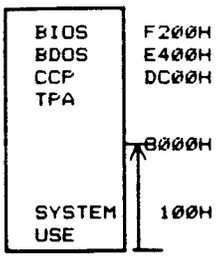


FIGURE 1.

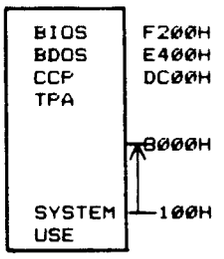


FIGURE 2.

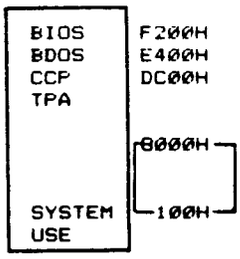


FIGURE 3.

Design Process

The previous paragraphs described the design requirements from a conceptual point of view. Essentially, the phantom* line will be used to enable an alternate memory bank only when requested by the memory management software in the CBIOS. This alternate memory segment will be addressed in the range of 100H to 8000H. Figure 4 is a block diagram that depicts the implementation of this statement with more detail. The diagram also shows the required input and outputs. There are many ways to implement the logic required to perform the functions contained within each block. Figure 5 is the actual logic diagram used in this particular example.

Theory Of Operation

The memory portion of the board is found in U1 through U6. 6116s were chosen because of the ease of implementation and their interchangeability with 2716 EPROMs. U7 is a 74LS138 used in a conventional memory decoder/chip select circuit. U8 is the port decoder and it utilizes the 74LS682. This particular IC (integrated circuit) was chosen because it contains the internal pullup resistors on the DIP switch inputs, thereby reducing the number of parts on the board. U9 performs the qualifier function and only permits the phantom* line to operate in the 100H to 8000H range. U12 is the control logic that determines if the

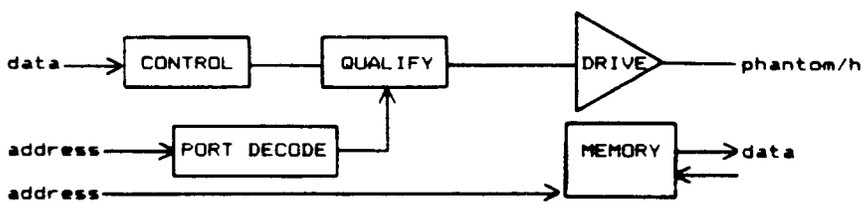


FIGURE 4.

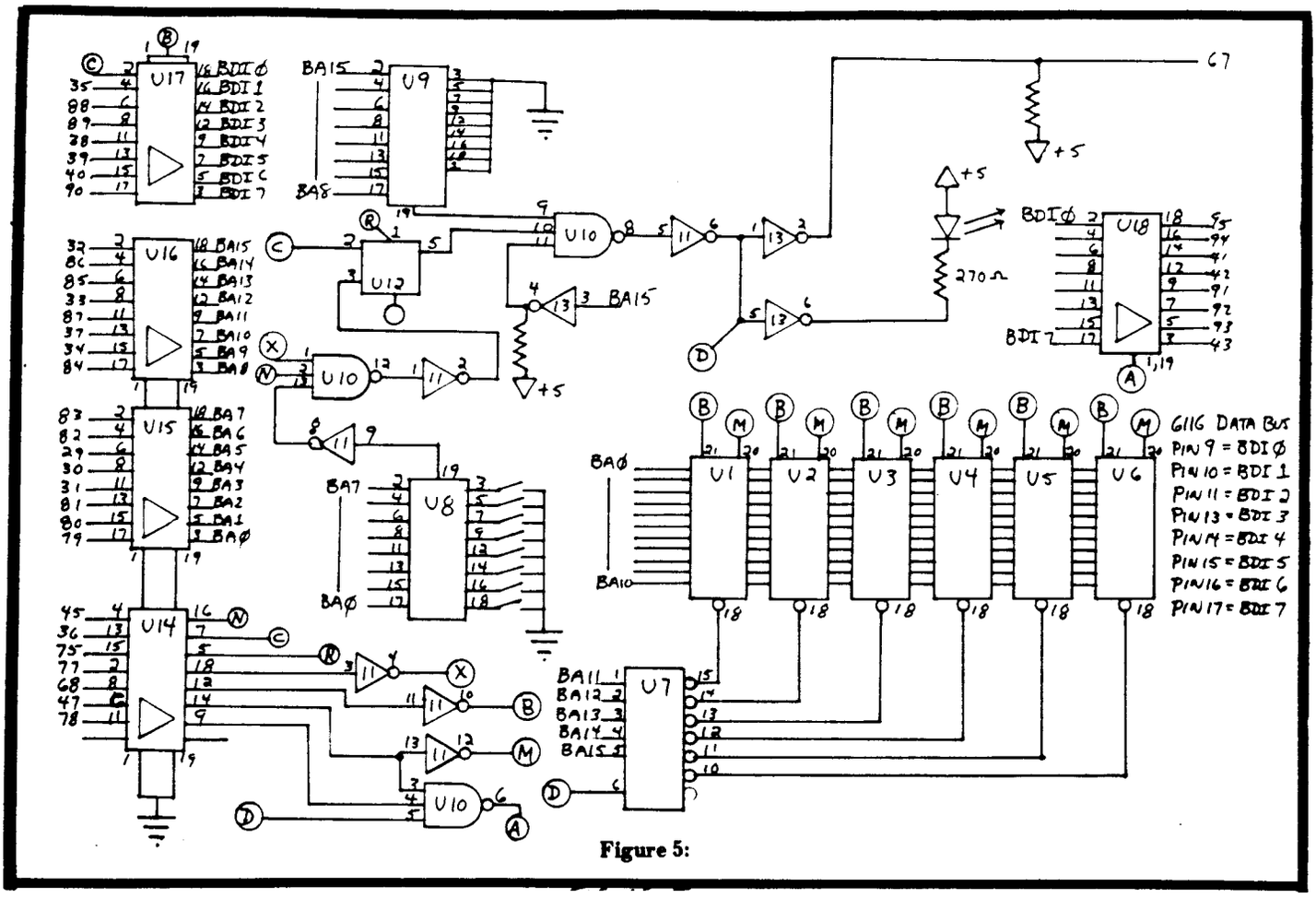


Figure 5:

alternate memory bank will be used in place of the user TPA. U14 thru U18 are drivers used to power all the devices on the board.

Construction

The materials shown on the parts list are "off the shelf" components and can be purchased from many different sup-

```
REQUIRED PARTS LIST:
        S100 prototype board SUN 721
             from SUNTRONICS CO.
        U1 thru U6 HM6116-4 200 nsec
        U7 74LS138
        U8 74LS682
        U9 74LS688
        U10 74LS10
        U11 74LS04
        U12 74LS74
        U13 7406
        U14 thru U18 74LS244
```

pliers. The construction techniques are not any more critical than those used in other projects involving LSTTL logic. There have been several articles in *The Computer Journal* on techniques used for prototyping, and this information could be used to build the board. Figure 6 is the parts layout that I used for this project—it functioned quite satisfactorily from a construction point of view.

Procedure 1 gives the syntax necessary to test the completed project using Digital Research's DDT (Dynamic Debug Tool). The upper limit used in the fill command should be set to the size of your alternate TPA. The value of 3000H corresponds to a 12K alternate TPA. Port address 90H was used to switch the TPAs in this example.

```
DDT                                     <ENTER>
A 9000                                  <ENTER>
9000        MVI A,01                    <ENTER>
9002        OUT 90                      <ENTER>
9004        NOP                         <ENTER>
9005                                    <ENTER>
0F000,9004                              <ENTER>
F100,3000,FF                            <ENTER>
D100                                    <ENTER>
D                                       <ENTER>
D can be entered as many times as necessary to check
  your particular configuration for all bits set.
F100,3000,00                            <ENTER>
D100                                    <ENTER>
D                                       <ENTER>
D can be entered as many times as necessary to check
  your particular configuration for all bits reset.

                PROCEDURE 1.
```

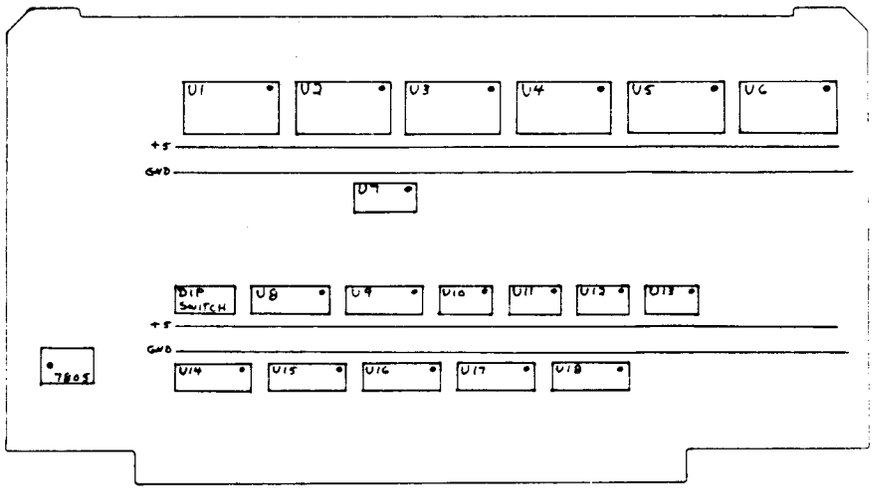
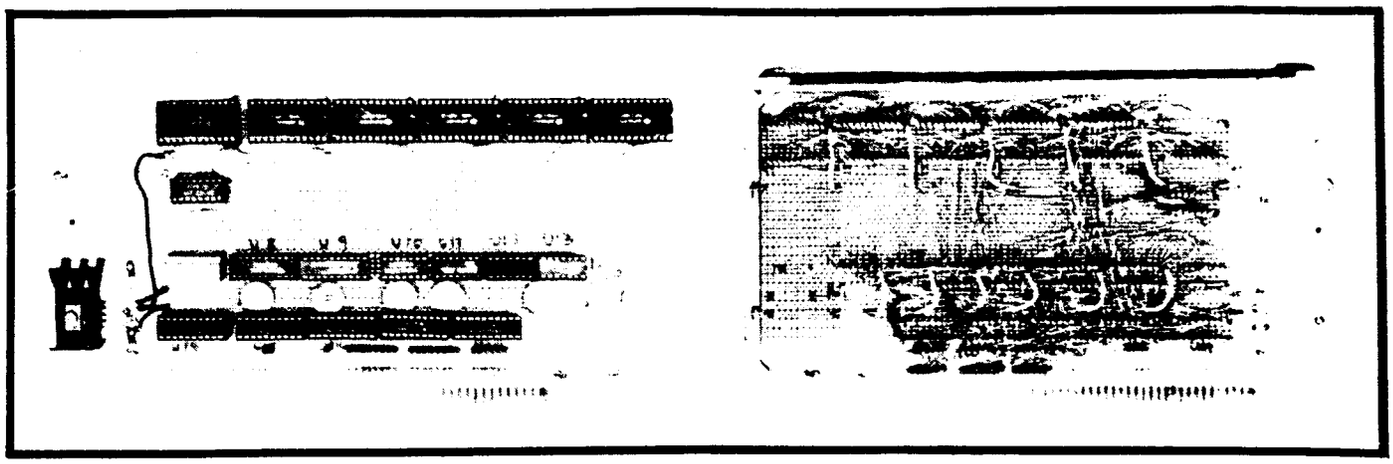


Figure 6:



Software Implementation

The description that follows is general in nature due to the many variations that exist in the CBIOS portions of CP/M system software. The first thing that has to be done is to determine what sections of the CBIOS have to be available to all users. These portions of the code will have to be maintained in the non-banked area of memory that was formerly used to contain the entire CBIOS. The portions of code that are only used by the extended CBIOS can be located in the banked area of memory. Typically, the non-banked area of memory will contain the "jump vector table," disk parameter blocks, disk parameter tables, the move data portion of disk read/write, and the memory management code to switch in/out of the banked memory. The banked area of memory can contain the system-specific drivers for all the I/O devices. This is the place to harvest the fruit of your labors in the implementation of this project. You can implement keyboard translation, I/O drivers for the reader/punch devices, additional drivers for the user list devices, full implementation of the I/O byte and drivers for a hard disk controller. You can do all of these things and more while maintaining a very large user TPA. Figure 7 is a listing which includes the non banked code of my CBIOS as implemented with this concept. The non banked segment begins with the BIOSOVL label. Each jump vector will jump to a segment of code that will arrange for a new temporary stack, switch TPAs and call the appropriate routine in the alternate TPA. The WBRET label is used as a return from the warm boot routine instead of a normal return from a call in-

Figure 7: Listing

```
;
;       *       MACROS FOR Z80 INSTRUCTIONS
        SSPD    MACRO   ADDR
                DB      0EDH,73H
                DW      ADDR
                ENDM
        LSPD    MACRO   ADDR1
                DB      0EDH,7BH
                DW      ADDR1
                ENDM
        LDIR    MACRO
                DB      0EDH,0B0H
                ENDM
        DJNZ    MACRO   ADDR2
                DB      10H
                DB      ADDR2-$-1
                ENDM
        SPLR    MACRO
                DB      0CBH,3FH
                ENDM
        SLAR    MACRO
                DB      0CBH,27H
                ENDM
;       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
                ASEG
                ORG     100H    ;ORIGIN OF THIS PROGRAM
;
;       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;
;
START:  LXI     H,START
        LXI     D,8100H
        LXI     B,3000H
        LDIR
        JMP     CONT+0000H
CONT:   MVI     A,01
        OUT     SHAPTPA
        LXI     H,8100H
        LXI     D,START
        LXI     B,3000H
        LDIR
        LXI     H,BIOSOVL
        LXI     D,BIOS
        LXI     B,3F00H
        LDIR
        MVI     A,00
        OUT     SHAPTPA
        OUT     RESETHD
         JMP    CCP+3
;
;
THIS AREA WILL CONTAIN THE BULK OF YOUR BIOS. THIS PORTION
OF THE BIOS WILL RESIDE IN THE BANKED RAM AREA.
;
;*---------------------------------
;CODE THAT FOLLOWS IS NEW BIOS VECTORS WITH TPA SWITCHING
;*---------------------------------
BIOSOVL         EQU     $
                .PHASE  0F200H
SCBOOT:         JMP     XSCBOOT
SMBOOT:         JMP     XSMBOOT
SCONST:         JMP     XSCONST
SCONIN:         JMP     XSCONIN
SCONOUT:        JMP     XSCONOUT
SLIST:          JMP     XSLIST
SPUNCH:         JMP     XSPUNCH
SREADER:        JMP     XSRDR
SHOME:          JMP     XSHOME
SSELDSK:        JMP     XSSELDSK
SSETTRK:        JMP     XSSETTRK
SSETSEC:        JMP     XSSETSEC
SSETDMA:        JMP     CSETDMA         ;GO DIRECT TO ROM
SREAD:          JMP     XSREAD
SWRITE:         JMP     XSWRITE
SLISTST:        JMP     XSLISTST
SSECTRN:        JMP     XSSECTRN
XSCBOOT:        LXI     SP,TEMPSTK
                CALL    NEXTPA
                JMP     BOOT
XSMBOOT:        LXI     SP,TEMPSTK
                CALL    NEXTPA
                JMP     MBOOT
MBRET:          CALL    OLDTPA
                JMP     CCP+3
XSCONST:        SSPD    STAKSAV
                LXI     SP,TEMPSTK
                CALL    NEXTPA
                CALL    CONST
                CALL    OLDTPA
                LSPD    STAKSAV
                RET
XSCONIN:        SSPD    STAKSAV
                LXI     SP,TEMPSTK
                CALL    NEXTPA
                CALL    CONIN
                CALL    OLDTPA
                LSPD    STAKSAV
                RET
XSCONOUT:       SSPD    STAKSAV
                LXI     SP,TEMPSTK
                CALL    NEXTPA
                CALL    CONOUT
                CALL    OLDTPA
                LSPD    STAKSAV
                RET
XSLIST:         SSPD    STAKSAV
                LXI     SP,TEMPSTK
                CALL    NEXTPA
                CALL    LIST
```

C Sick?

PLZ is the cure!

Introducing a native code PLZ compiler for the 68000, featuring:

- ☐ Complete PLZ language, including structure assignment and comparison
- ☐ Fully compatible with Zilog Z80, Z8000 PLZ
- ☐ Ideal for embedded, ROM based systems
- ☐ Strongly typed
- ☐ Data types include signed and unsigned byte, word and longword
- ☐ All of the protection of Pascal, with the flexibility of C
- ☐ Inherently more portable than either Pascal or C
- ☐ Easy for Pascal or C programmers to learn
- ☐ Fully compatible with the CP/M-68K C library

Requires CP/M-68K. Other systems and CPU's supported soon

Package includes:
68000 Compiler-Code generator
User Manual
Springer-Verlag "Report on the Programming Language PLZ/SYS"
One Year free updates

All this for the low introductory price of $75

Add $3 S/H. NJ Residents include 6% sales tax

KCSystems
20 Lamington Drive, Succasunna, NJ 07876
(201) 927-9104

APROTEK 1000™ EPROM PROGRAMMER

only $250.00

TECHNICAL BREAKTHROUGH NOW ALLOWS A PRICE BREAKTHROUGH

A SIMPLE, INEXPENSIVE SOLUTION TO PROGRAMMING EPROMS

The ***APROTEK 1000*** can program 5 volt 25XX series through 2564, 27XX series through 27256 and 68XX devices plus any CMOS versions of the above types. Included with each programmer is a personality module of your choice (others are only $10.00 ea. when purchased with ***APROTEK 1000***). Later, you may require future modules at only $15.00 ea., postage paid. Available personality modules: PM2716, PM2732, PM2732A, PM2764, PM2764A, PM27128, PM27256, PM2532, PM2564, PM68764 (includes 68766). (Please specify modules by these numbers)

APROTEK 1000 comes complete with a menu driven BASIC driver programmer listing which allows READ, WRITE, COPY, and VERIFY with Checksum. Easily adapted for use with IBM, Apple, Kaypro, and other microcomputers with a RS-232 port. Also included is a menu driven CPM assembly language driver listing with Z-80 (DART) and 8080 (8251) I/O port examples. Interface is a simple 3-wire RS-232C with a female DB-25 connector. A handshake character is sent by the programmer after programming each byte. The interface is switch selectable at the following 6 baud rates: 300, 1.2k, 2.4k, 4.8k, 9.6k and 19.2k baud. Data format for programming is "absolute code" (i.e., it will program exactly what it is sent starting at EPROM address 0). Other standard downloading formats are easily converted to absolute (object) code.

The ***APROTEK 1000*** is truly universal. It comes standard at 117 VAC 50-60 HZ and may be internally jumpered for 220-240 VAC 50-60 AZ. FCC verification (CLASS B) has been obtained for the ***APROTEK 1000***.

APROTEK 1000 is covered by a 1 year parts and labor warranty.

FINALLY — A Simple, Inexpensive Solution To Erasing EPROMS

APROTEK-200™ EPROM ERASER
Simply insert one or two EPROMS and switch ON. In about 10 minutes, you switch OFF and are ready to reprogram.
APROTEK-200™ only ***$45.00.***

APROTEK-300™ only ***$60.00.***
This eraser is identical to ***APROTEK 200™*** but has a built-in timer so that the ultraviolet lamp automatically turns off in 10 minutes, eliminating any risk of overexposure damage to your EPROMS.
APROTEK-300™ only ***$60.00.***

APROPOS TECHNOLOGY
1071-A Avenida Acaso, Camarillo, CA 93010
CALL OUR TOLL FREE ORDER LINES TODAY:
1-(800) 962-5800 USA or 1-(800) 962-3800 CALIFORNIA
TECHNICAL INFORMATION: 1-(805) 482-3604
Add Shipping Per Item: $3.00 Cont. U.S. $6.00 CAN, Mexico, HI, AK, UPS Blue

PERSONAL ROBOTICS EXCLUSIVELY
FACTORY AUTHORIZED DEALERS

HAVE YOU HEARD ABOUT SAVVY?
SAVVY is a revolutionary programming language for the Apple II+ & IIe and the IBM PC. It approaches artificial intelligence. Besides being a perfect language for robot control, it can handle general applications programming. Write or call for detailed descriptive material.

A FULL-LINE ROBOT SUPPLIER
1. No dissatisfied customers
2. Call if you have questions—we know our products
3. Unlimited help when you need it

KITS	List	SALE
MOVIT		
Screwdriver Kits—No Solder—FUN		
Avoider	44.95	40.15
Circular	67.95	63.75
Line Tracer	39.95	36.75
Medusa	27.95	26.25
Memoconcrawler	74.95	70.15
Monkey	24.95	23.65
Mr. Bootsman	30.95	29.25
Peppy	24.95	23.65
Piper Mouse	44.95	40.13
Sound Skipper	24.95	23.65
Turn Backer	39.95	36.75
We have sold hundreds of these items. They work as they should and they last.		
HARVARD ASSOC.		
Turtle Tot	299.00	275.00
Excellent for simple educational applications. Our customers have been very satisfied.		
ROBOT SHOP		
Robot Bug	129.95	122.00
Z-1	149.95	136.50
Z-2	249.95	223.50
Simple and dumb, but lots of potential for customizing and expansion		

ROBOTS	List	SALE
ANDROBOT		
Slave to your computer		
TOPO III	1595.00	1485.00
Accessories		CALL
TOPO III moves and talks very well. It uses LOGO-like commands and a good text-to-speech system		
RB ROBOT		
Download from your computer		
RB5X	2295.00	2058.00
Accessories		CALL
Program it in Tiny BASIC or use a SAVVY to Tiny BASIC "compiler". RB5X has bumpers and sonar, voice with sound effects, arm, easy-to-use programming language, additional senses are available. It is good quality. Our two demos have seen hard use and haven't broken down yet!		
ARCTEC		
GEMINI	6995.00	6506.00
A marvel. Three CPUs, voice recognition, goal-oriented navigation, on-board disk drive, detachable keyboard, and more		

TECHNICAL INFO—WE CAN HELP. CALL

Shipping Over $200 Add 4%. $200 And Under Add 5%
Cash With Order Deduct 3%. N.M. Orders Add 5% Sales Tax
Allow 4 Weeks For Delivery

RIO GRANDE ROBOTICS
A Division of Mobile Intelligence Corporation
1595 W. Picacho #28, Las Cruces, N.M. 88005, Tel. (505) 524-9480

CTM

"...received my moneys worth with just one issue..."
—J. Trenbick

"...always stop to read CTM, even though most other magazines I receive (and write for) only get cursory examination..."
—Fred Blechman, K6UGT

U.S.A.	$15.00 for 1 year
Mexico, Canada	$25.00
Foreign	$35.00(land) · $55.00(air)
(U.S. funds only)	
Permanent (U.S. Subscription)	$100.00
Sample Copy	$3.50

CHET LAMBERT, W4WDR
1704 Sam Drive • Birmingham, AL 35235
(205) 854-0271

```
                CALL    OLDTPA
                LSPD    STAKSAV
                RET
XSPUNCH:        SSPD    STAKSAV
                LXI     SP,TEMPSTK
                CALL    NEXTPA
                CALL    PUNCH
                CALL    OLDTPA
                LSPD    STAKSAV
                RET
XSRDR:          SSPD    STAKSAV
                LXI     SP,TEMPSTK
                CALL    NEXTPA
                CALL    READER
                CALL    OLDTPA
                LSPD    STAKSAV
                RET
XSHOME:         SSPD    STAKSAV
                LXI     SP,TEMPSTK
                CALL    NEXTPA
                CALL    HOME
                CALL    OLDTPA
                LSPD    STAKSAV
                RET
XSSELDSK:       SSPD    STAKSAV
                LXI     SP,TEMPSTK
                CALL    NEXTPA
                CALL    SELDSK
                CALL    OLDTPA
                LSPD    STAKSAV
                RET
XSSETTRK:       SSPD    STAKSAV
                LXI     SP,TEMPSTK
                CALL    NEXTPA
                CALL    SETTRK
                CALL    OLDTPA
                LSPD    STAKSAV
                RET
XSSETSEC:       SSPD    STAKSAV
                LXI     SP,TEMPSTK
                CALL    NEXTPA
                CALL    SETSEC
                CALL    OLDTPA
                LSPD    STAKSAV
                RET
XSREAD:         SSPD    STAKSAV
                LXI     SP,TEMPSTK
                CALL    NEXTPA
                CALL    READ
                CALL    OLDTPA
                LSPD    STAKSAV
                RET
XSWRITE:        SSPD    STAKSAV
                LXI     SP,TEMPSTK
                CALL    NEXTPA

                CALL    WRITE
                CALL    OLDTPA
                LSPD    STAKSAV
                RET
XSLISTST:       SSPD    STAKSAV
                LXI     SP,TEMPSTK
                CALL    NEXTPA
                CALL    LISTST
                CALL    OLDTPA
                LSPD    STAKSAV
                RET
XSSECTRN:       SSPD    STAKSAV
                LXI     SP,TEMPSTK
                CALL    NEXTPA
                CALL    SECTRAN
                CALL    OLDTPA
                LSPD    STAKSAV
                RET
NEXTPA:         PUSH    PSW
                MVI     A,01
                OUT     SWAPTPA
                POP     PSW
                RET
OLDTPA:         PUSH    PSW
                MVI     A,00
                OUT     SWAPTPA
                POP     PSW
                RET
DISKRDWR:       CALL    OLDTPA
RNEXT:          CALL    DISKREAD
                CALL    NEXTPA
                RET
MVEDMA:
                CALL    OLDTPA
                LDIR
                CALL    NEXTPA
                RET

;
;       X X X
```

A $500 SUPERBRAIN COMPUTER

by Bill Kibler

With prices of computers coming down, there are many inexpensive used units on the market. My local used computer store had a unit that was not moving, and looked too good to pass up. It was a Superbrain QD by Intertec Data Systems. I remembered the name from working at Micropro, where they configured Wordstar for it. The programmers liked some things but had complaints about others—rather typical, I thought. Because my fellow workers didn't give it a completely negative report, I felt a low bid might get me a good second computer. This unit had cost over $2000 new and was on sale for $795, marked down from $995. I said "$500 or forget it," and found myself walking out with it.

Although I am using the Superbrain to write this article, there are times that I feel I spent too much money. As I explain more about the unit you will see why this is a good but limited unit. One of the reasons for the low price was the lack of documentation. The owner did supply a bound manual after the purchase, which only proved my choice of pricing.

Intertec Support?

Remembering that Intertec advertised in *BYTE*, I looked up their phone number and gave them a call. They do not support the product. The company has turned support over to three separate companies located on the East Coast. This situation is recent, and only one place talked like they actually were supporting the unit. The other two were contract servicing people trying to cut out a national position for themselves. Actually, I feel that Intertec just dumped it on them with a "take it or else."

So what kind of support can you get? Not much. Manuals go for $50 to $75 each, which is a little high for copies of old CP/M manuals. The schematics also go for $50 to $100 a set and contain drawings for several versions of the Superbrain. One feature of the Superbrain was to have been a single S-100 slot. The factory never made it work, but one of the support places has it for $500. A parallel interface card to hook Superbrain units together in a net system is also available. I feel that this net system is why the company stayed alive for so long.

Since I needed more information on the unit and did not want to pay for schematics, I tried several local places. One place here in California handles these units and gave me copies of the schematics for free. From them I found out that the unit is used mostly by the government. The net ability was one of the first on the market, and so governments bought them in large quantities. Intertec is also trying to build an IBM type unit, and probably decided that supporting its old units was taking away from the development of new products. If this had been their only failure, I might have a better feeling towards them.

BIOS Mainia

This is where the fun starts. Several of my needs were for 8 inch compatibility and Centronics printer operation. The unit has two 5 inch drives of the old double sided 35 track variety. Running their 3.1 version of BIOS gives 380K with 512 bytes per sector, which is not bad. Two serial ports, which are easily changed in a configuration program, talk to printers or modems. The unit has two Z-80s and 64K of memory, with memory mapped video and a built in keyboard. It is possible to see that some imagination

```
**************************************************

MEMORY MAP  (EQUIVALENT SYSTEM SIZE OF 57K)

F800 TO FFFF   CRT MEMORY MAP
F600 TO F7FF   DISK HOST BUFFER
F400 TO F5FF   * NOTHING *
F380 TO F3FF   STACKS
F300 TO F37F   KEYBOARD BUFFER
F280 TO F2FF   DIRECTORY BUFFER
EF20 TO F27F   * NOTHING *
EF00 TO EF1F   CONFIGURATION DATA
E800 TO EEFF   WARM START ROUTINES
EBA8 TO EE7F   * NOTHING * (COPY OF BDOS)
EAD4 TO EBA7   DISK HANDLING ROUTINES
E7C6 TO EAD3   CRT OUTPUT ROUTINES
E713 TO E7C5   CRT INPUT ROUTINES
E483 TO E712   INTERRUPTS AND INITIALIZATION
E400 TO E482   JUMP TABLES AND BUFFERS
E293 TO E3FF   * NOTHING * (BIOS FREE SPACE)
DE00 TO E292   BIOS "QD31BIOS.ASM"
D000 TO DDFF   CP/M BDOS
C800 TO CFFF   CP/M CCP
0100 TO C7FF   TPA (~50K)
0000 TO 00FF   BASE PAGE 0, CPM JUMPS

**************************************************
```

went into the hardware design of the unit, but not so for the software.

I printed out the listing of the BIOS and found references to other jump tables. After DDT-ing around in the system area of memory for awhile, I decided that only one third of the listing was provided. What the other two thirds of the BIOS and the BOOT PROM contained was a mystery. A long distance call to Intertec gave the fateful answer "that information is not available to anybody," and that includes the support companies as well. So out came the disassembler, and several days later I had the reason why.

I have never seen such a mess of code in my life. Either the programmer left it that way to keep others out, or somebody was in a big hurry. It appears that the last programmers patched everything. The jumps out of the BIOS go to jump tables that jump to the actual operations. It is possible that different people were patching different areas and so they decided that one place would have the jumps to their individually written routines. Whatever the reasoning, it is almost impossible to change the system. There is some space to lengthen the BIOS, but any major changes will require decoding other routines first. I might also add that past code has been left in the system, such as parts of the BDOS. Routines that were later not used are still there too.

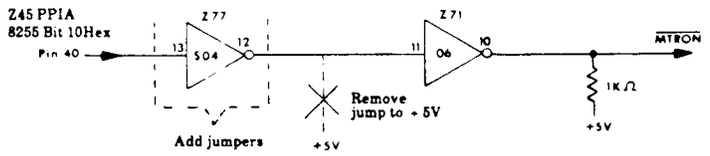
The use of ZSID and Disassembler will be necessary to see where real code is and is not. Make sure your jumps go to routines, as well as routines actually being used (the disk routine has unused but valid code). If time permits, I will rewrite the BIOS to include all their routines, or better ones, and put it in the public domain. Because theirs is such a mess, and contains bad and bogus code, a new version will be at least 2K less in length.

Modifications

A later version of the software turned the drives off after use; mine does not. The parallel chip that provides internal commands has several extra outputs. My first choice killed the system, but the second one worked fine. The routines in the listing show where to patch the BIOS to get the drives to turn off. This is not the best way to do it, but without access to the interrupt routines (clock functions), a proper or normal modification is not possible. I am running this mod right now and have not found any problems yet.

The 8 second delay timer is necessary to keep the drives from cycling during format or multiple disk copying. I tried everything to avoid using this circuit, since I have a real dislike of adding transistors to logic circuits. The diagram in Figure 1 however, is the only one that would do the job for me with the least amount of chips and hardware. Also shown is the circuit without the timer, which will work, but it causes the drives to cycle too much. You might try the simple jumpers first and then add the 8 second timer. I put everthing on one chip socket and then glued it to the main board for a rather quick and simple installation.

Initial Hookup



8 Second Delay Circuit

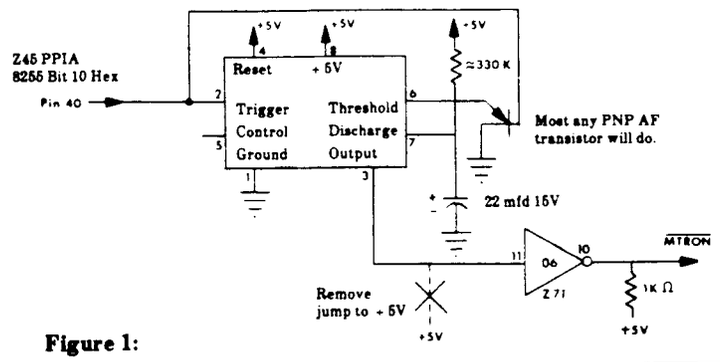


Figure 1:

Disk Formats

While using a friend's Kaypro, I tried to make disks on it for the Superbrain.

```
**************************************************
;
; SID AND ZSID PATCHES DONE WITH DDT
;
;    SID.COM
08F2       MVI  A,C3
08F4       STA  0038h  ;change to 0030hex, rst 6
08F7       LXI  h,0686h
08FA       SHLD 0039h  ;change to 0031hex
;
;
;    ZSID.COM
10FD       MVI  A,C3
10FF       STA  0038 ;change to 0030hex, rst 6
1102       LXI  H,0E86
1105       SHLD 0039 ;change to 0031hex
;
;
**************************************************
```

Three known support organizations:

A) Kramer Systems International
(301) 933-8300

B) Molenair Corporation
(800) 328-8944

C) Nine Associates
(703) 273-1803

Although formats were given for the Superbrain, they were not compatible with my system. The directory is on the wrong track, which has made it non-compatible so far. Getting data to and from the unit requires Modem 7. The PIP program has been patched for machine to machine transfers. Unfortunately, PIP cannot transfer files larger than 16K in this manner. You will find, however, that MBOOT can be transferred, reassembled, and then used to load Modem 7 files. There was another program without any documentation that I believe was for file transferring. The SEND and RECV programs never worked for me, while Modem 7 works just fine, especially at 9600 Baud. Watch out for the handshaking on the serial ports; their main port is not standard, and an adapter with 2 and 3 reversed but all others straight through may be needed.

One major problem is the INTERRUPT system. Superbrain uses Mode 1 interrupts, which duplicate the 8080's interrupt structure. These interrupts use the first 40Hex addresses for jumps when interrupted. My first encounter with this was trying to run SID or ZSID, the enhanced versions of DDT. They use RST 7 (restart 7) which jumps to 38Hex whenever it appears in the program. So if you try using SID, it will lock the system up. SID and ZSID can be patched, but when I used RST 6, some erratic operation was noted. Kaypro's have the same structure, so look for information in Kaypro magazines for help in this area. I have listed the patches for both programs to help you get started, but I guarantee nothing!

```
SOFTWARE CHANGES FOR MOTOR ON/OFF

INSERT ROUTINES CALLS TO "DRON" AND "DROFF"
;
HOME:       ..
            ..
            MOV B,A
            call dron       ;turn motor on
            CALL DISK
            call droff      ;turn motor off
            ORA  A
            ..
            ..
;
WRIT1:      call dron       ;turn motor on
            CALL DISK
            call droff      ;
            ORA  A
            ..
            ..
;
READHST:    CALL PARM
            MVI  B,1
            call dron       ;turn motor on
            CALL DISK
            call droff      ;turn motor off
            ORA  A
            ..
            ..
;
; ADD THESE TWO ROUTINES
;
dron:       push psw
            in   ppia       ;get data port 68h
;            set  4,a        ;turn on bits
            db   0cbh,0a7h  ;z80 opcodes
            out  ppia       ;turn on motors
            pop  psw
            ret
;
droff:      push psw
            in   ppia       ;get data port 68h
;            res  4,a        ;turn off bits
            db   0cbh,0e7h  ;z80 opcodes
            out  ppia       ;turn it off
            pop psw
            ret
;
```

Conclusion

This system has many drawbacks, and with time I will get a new BIOS that I can modify and reassemble. I see this, and the small TPA, as the main drawbacks. The few changes have made the motors turn off, and the return to a quiet-running system is a pleasure. (I hated the fan noise of my Z-100 too!) The unit's size is compact — not as compact as some portables, but about the same weight. I have found myself relying on this system a lot, and it is beginning to look like it may be a permanent part of my computing operation.

My final conclusion on this unit would be to buy. By that I mean that the unit at or near $500 is an excellent purchase if your demands are for a straightforward system and you can stand the limited TPA. The best use of the unit is probably for wordprocessing, especially with its memory mapped CRT. The screen does not have true descenders, but the update of the screen is so fast that writing is quite a pleasure. ■

BASE

A Series on How To Design and Write Your Own Database

By E.G. Brooner

Wrapping It All Up

In earlier installments, BASE was presented in sections with some explanation of how the program was conceived and put together. The last article in the series attempted to answer some of the questions that have come up as the program was developed and presented. In this issue we'll present an overview of the project with some comments that may help the potential database writer.

Memory and Storage Space

First of all, those of us who program realize that there is no do-all program of any genre, and that if there were such an animal it would be impractically large and unwieldy. I still remember trying to write useable programs with 16K of memory and one single sided single density disk drive. I won't even mention what it was like before that. Consequently, we always have to be aware of the limits of memory and storage space. These considerations are perhaps the ultimate constraint on any such project.

The program as presented so far occupies 18K of memory if compiled with CB-80. It is slightly smaller if compiled with C-BAS but then you have to load, in addition, the C-RUN utility – so there is no gain there. Some arrays are generated and they, too, take up space. If we arbitrarily limit the program to handling 1000 records, the arrays can consume (depending on the size of the fields being stored in them) 20K or more. At this point we are approaching the limit of a 48K CP/M system. Consequently, some of the features were made as separate programs that have to be chained by the main program. As shown so far, these are FILESORT (which sorts any file on any field) and PRTFORM which is a way of generating special report formats. It should be emphasized that neither of these has yet been included in this series of articles. They can easily be omitted, because all necessary finding and printing can be done, by BASE, from the raw data files.

Two additional, separate programs have been generated for my own use; one of these enables the matching of two key fields for a more selective search, and the other generates a file for making binary searches. It is my intention to combine all of these eventually, keeping in mind that this will strain my own 56K system to the limit.

"One of the definitions of a database is it lets you have some control without writing new code each time."

What Is It Good For?

Many people ask about databases (as we used to about microcomputers) "What do you intend to do with it?" I suppose 'store and retrieve data' is an inadequate answer. As was the case with micros themselves, you don't know until you have one. Most programs deal with data in some manner. One of the definitions of a database is that it lets you have some control over the process without writing new code each time your requirement changes. First of all, you can design the way the data will be stored; more importantly, you specify how it will be retrieved.

My main use for this kind of package is the retrieval of *subsets* of data. In the case of names, the subset might be everyone with the last name of Smith. 'Smith' would be the search key. Along with the Smiths would come all of their addresses, phone numbers, and whatever else I had associated with them. With numerical data it might be all entries above 100, or all between 50 and 100. In the case of snowfall records it could be an identification of the days on which snowfall exceeded a certain minimum limit.

Data is generally retrieved in one of two ways: as a subset (even a subset of one record meeting some 'key' criteria, and by an absolutely unique key. For unique key retrieval there can be one and only one answer. 'Smith' will not do. It must be J.P. Smith, for example, or even the Smith who lives in Podunk, North Dakota.

How The Searches Can Be Used

Any subset requires inspection of the entire file, hence sequential reading of an un-sorted file is about as good a way as any. In the case of BASE, the subset can be defined 1) as equal to a particular key (which can be only part of the key as SM for Smith), 2) by the greater than or less than comparison, and 3) by all greater than one key and less than another.

The two best ways to find a unique key are hashing and binary searching. Hashing is slightly faster but less flexible – the separate program that was mentioned earlier, therefore, deals with binary searching. The methods just mentioned handle all of my present requirements; indeed, I could do without the binary search except that it is so impressive and was so much fun to write. We mentioned, also, matching two keys within one record. This has proven useful even though it, too, was implemented as a separate program. For most purposes you can search for a subset and isolate the one you want by inspection of the very few listings that result.

Assume that you are a collector of both modern and antique guns, and want to create a database to serve as an inventory of your collection. The use of subsets and other database features would justify putting such an inventory on a database even though it involved only 50 or 100 pieces. The items of interest might be: make, model, serial number, source, cost, caliber, and several comment lines.

If one of these happened to be a Winchester rifle, model 1886, 44-40, it might be part of the following subsets: Winchester, rifle, and 44-40. You might have other Winchester guns, rifles of other makes, and (among the antiques) a 44-40 pistol. That particular item

would be part of each subset. You might also want to match two fields and list part of your collection as 'Winchester – Rifle(s).'

Perhaps the only unique characteristic – one that could identify it specifically in isolation from all other items – would be the serial number. You would probably never search this particular file by a comment field, but you might, depending on how you coded your comments. One or more comment fields is a good idea in any database to allow for later additions of information you hadn't anticipated. The possible structure might be designed as shown in Figure 1.

Keep in mind that you could have had up to 12 fields in this record, that each field is named by you and its length is specified when the base is first designed and created. This set of records need bear no relation to any other data collections you may have, and all can be run from the same program.

Similarly, a name-and-address file structure might be designed as follows:

1	F. Name	20 char
3	Addr 1	20 char
4	Addr 2	20 Char
5	ZIP	9 char
6	Phone	12 char
7	Business	15 char
8	Holiday	5 char
9	Comment 1	20 char
10	Comment 2	10 char

A file such as this could be used for addressing, for phoning, for printing labels, or whatever. It is not necessary to use every field for every purpose. Too, you would have designed the file yourself (perhaps in a radically different way) to suit your own purpose.

Some Future Improvements

Some database applications can benefit from mathematical manipulation. We have not allowed for this in BASE; in fact, even numerical fields are stored as strings. If you have this need it will be necessary to find the field, then convert it (for example, AVAL (A$) before using it arithmetically. Numbers can still be compared, and sorted, while in string form. We plan to incorporate this feature into the report generator, PRTFORM.

We occasionally want to delete an existing record and reclaim the space it occupies. A simple way to do this is to first locate the record, then modify it by placing DELETE in the first field. When adding another record, then, first search for a DELETE and then use the 'modify' option to enter the new record over the old one.

Finally, we are in process of combining the present separate programs and the hoped-for improvements, all in one program, as a finished product. This version will be published if enough interest is expressed by readers and those field-testing the package.

Another desirable change involves the length of each field. The CREATE mocule (as shown) arbitrarily limits these to 20 characters; it has proven to be a bit short for some purposes. This limit can be changed easily at line 2200 but must be done before creating any files. There should, of course, be some limit to keep file sizes reasonable.

Field # and name		space allowed	contents
Field 1	Make	(20 char)	Winchester
2	Model	(10 char)	1886
3	Type	(10 char)	Rifle
4	Caliber	(6 char)	44-40
5	Serial #	(15 char)	12345WW
6	Value	(6 char)	$700
7	Comment	(20 char)	----------

Figure 1:

And, Miscellaneous Comments

The concatenation of file names by the program, and its generation of so many sub-files for each database you create, results in a great many directory entries. Many CP/M 80 systems are limited to 64 entries and this number can be approached quite rapidly. Should the user get near this number of files it would be wise to simply start another diskette for future data collections. In my own experience I have usually run out of directory space before filling the disk with data. ■

"BMON"

Software In-Circuit Emulator

Links your CP/M computer with any Z80 based computer or controller that you may develop. All that is needed is BMON, 8K of ROM space, and a handshakeable bi-directable I/O port (either RS232 or Parallel).

Features:

—Full program development debugger with Breakpoints, Snaps, Stops, & Waits.

—Single Step program execution.

—Download file from CP/M system to development RAM.

—Upload Memory from development RAM to CP/M disk.

—Two versions: Master BMON runs in your CP/M system, Slave BMON runs in your target system.

Note: Requires Microsoft's M80 & L80 assembler & linker to setup Slave BMON.

8" SSSD Disk containing Master BMON, Slave BMON, CONSOL, BMONIO, CONSOLIO, and Users Manual $49.95

Shipped Via prepaid UPS
—No COD or P.O. Box—
Check or Money Order to:

Barnes Research & Development
750 W. Ventura St.
Altadena, CA 91101
(818) 794-1244

CP/M is a trademark of Digital Research Inc
M80 & L80 are trademarks of Microsoft Inc

SOURCES WANTED

We need sources for the following items which are being used in projects for future articles. If you know where individuals can order these items at a reasonable price in small quantities, share the information with others.

Thermocouples
Strain gages
Flow meters
Pressure sensors

Drop a line to *The Computer Journal* with your suggestions on these, or to add other items to the list.

Interfacing Tips and Troubles

A Column by Neil Bungard

Last month we introduced the TONE CONTROL, a device which monitors the telephone's dual tone keypad frequencies. In that article we reviewed some touch tone telephone basics and discussed the TONE CONTROL's circuit operation. This month we will conclude the discussion of the TONE CONTROL by explaining how to use, construct, and tune the TONE CONTROL circuit.

Using the Tone Control

In order to allow the TONE CONTROL to monitor the phone line as discussed last month, incoming calls must be answered automatically. Although there are a number of circuits for accomplishing this task, the circuit in Figure 1 will do the job nicely. The advantage of this circuit is that the computer has control of hanging up the telephone. With this capability you can program the computer to consider a unique touch tone key specifically for that purpose. One of the disadvantages of this circuit is that the computer has no way of knowing when the telephone is off the hook. Consequently, if someone calls and does not instruct the computer (via the touch tone keypad) to hang up, the telephone may stay off the hook indefinitely. This problem can be eliminated with a software routine which periodically checks the line to determine if it is being used.

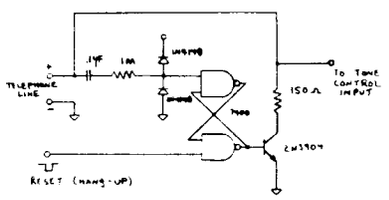


FIG. 1 ANSWERING CIRCUIT

The computer must generate one input timing signal and one output control signal for the TONE CONTROL's operation. The input timing signal is required to obtain the keypress information from the TONE CONTROL. This timing signal can also be used to determine the condition of the keypress valid flag by polling the tristate device (see Figure 2). When polling to detect this flag, you should input the contents of the octal tristate device approximately every 100 milliseconds. The keypress valid flag is present on bit 7 of the data byte and is input each time the 74LS373 is polled. After the 74LS373 is polled and determined to be a logic 0, a short time delay for filtering purposes is executed (about 15msec), then data bit 7 is rechecked. If data bit 7 is a logic 1, no keypress is available. If data bit 7 is a logic 0, a valid keypress is available, and that keypress information is contained in bits 1 through 6. Refer to Figure 3 for a flow diagram representing the software required to utilize this method.

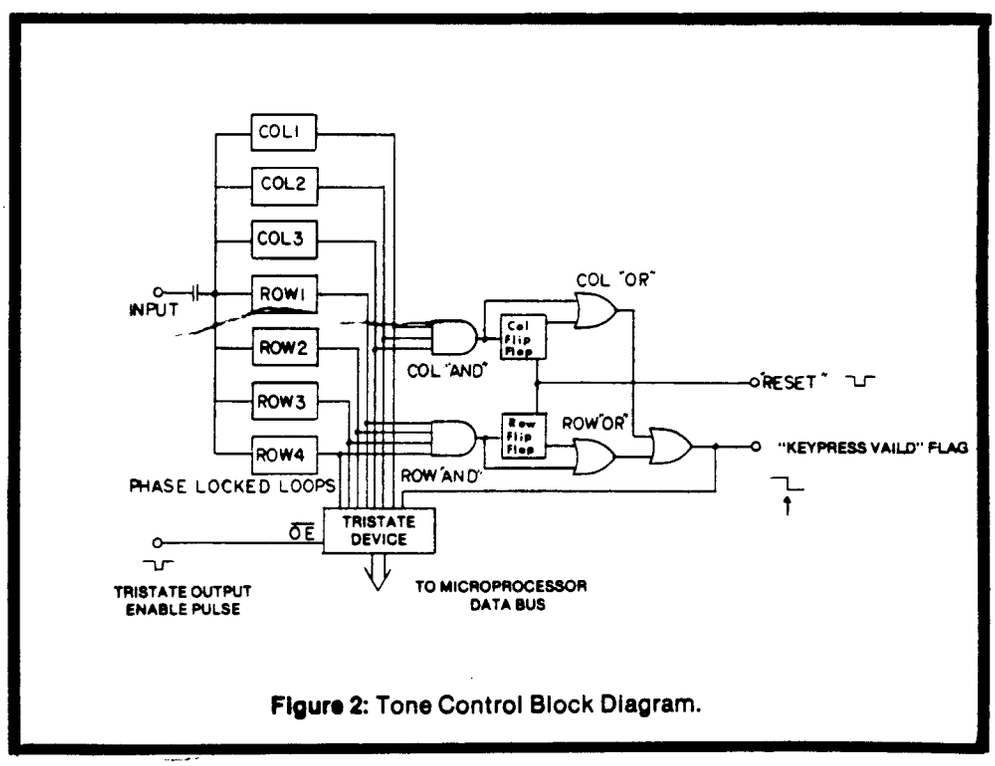


Figure 2: Tone Control Block Diagram.

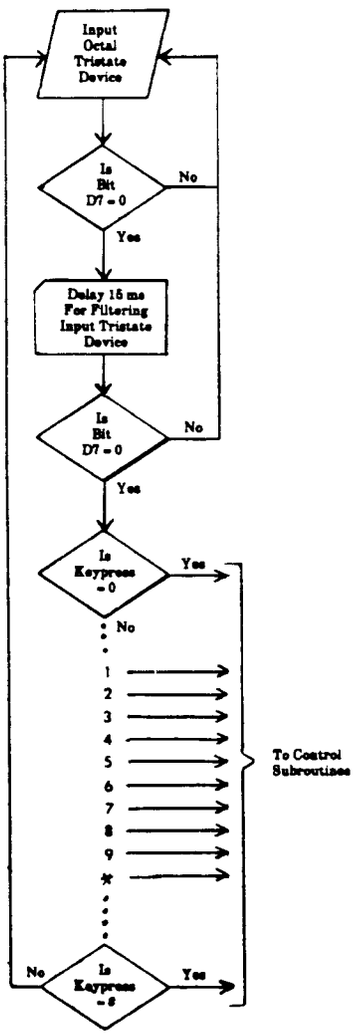


Figure 3: Polling Flowchart

An alternative method for determining a valid keypress is to connect the keypress valid flag to the interrupt input of the computer. When the keypress valid flag goes to a logic 0, the copmputer is alerted that a valid keypress may be awaiting input. When interrupted, the computer will go to an interrupt subroutine where the validity of the keypress will be determined. If the keypress is determined to be valid, the keypress information will then be input to the computer. Figure 4 shows a

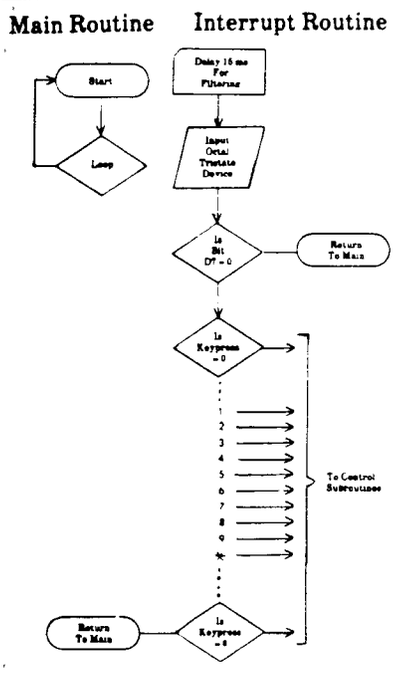


Figure 4: Interrupt Flowchart

flow diagram of the program required to utilize the interrupt method. Figure 5 shows the bit pattern generated for the corresponding touch tone keys, and also shows the hexadecimal representation of each key.

The output control signal required by the TONE CONTROL is used to reset the row and column flip flops. Resetting the flip flops informs the tone control that the current keypress has been input by the computer. Figure 6 shows a timing diagram of the signals required to detect a keypress, input the keypress, and reset the keypress valid flag.

Constructing and Tuning the TONE CONTROL

The TONE CONTROL circuit can either be wire wrapped or placed on a printed circuit board. Foil patterns for the TONE CONTROL circuit are shown in Figure 7. A parts placement diagram and schematic to aid in construction are shown in Figures 8 and 9 respectively. Be careful to observe proper polarity when placing the capacitors and the diodes on the PC board. Also, be sure that pin 1 on all ICs

Telephone Disit	Data Bus Interface D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	Hexidecimal
0	0 0 1 1 0 1 1 1	37
1	0 1 1 1 1 1 0 0	7C
2	0 1 1 1 0 1 1 0	76
3	0 1 0 1 1 1 1 0	57
4	0 1 1 1 1 0 0 1	79
5	0 1 1 1 0 0 1 1	73
6	0 1 0 1 1 0 1 1	5B
7	0 1 1 0 1 1 0 1	6D
8	0 1 1 0 0 1 1 1	67
9	0 1 0 0 1 1 1 1	4F
*	0 0 1 1 1 1 0 1	3D
#	0 0 0 1 1 1 1 1	1F

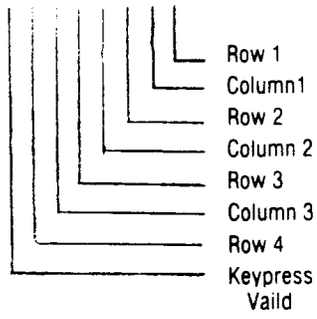


Figure 5:
Row and column bit pattern

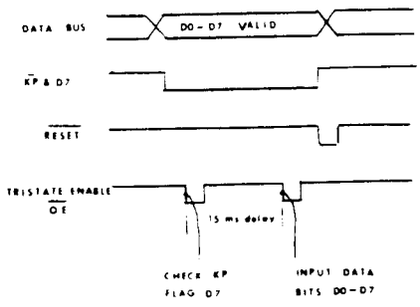


Figure 6: Timing diagram of detect keypress, input keypress, and reset 'KP' flag.

is in the correct position. Before using this device, each PLL (Phase Locked Loop) must be trimmed to a separate row or column frequency. In tuning a PLL, the values of four timing components should be considered. These components are R1, C1, C2, and C3 (see Figure 10). F0 is the desired PLL detection frequency. First a value for C1 was chosen. C1's limits can be between 0.01 microfarad and 1.0 microfarad.

Calculate R1:

$R1 = 1.1 + (f0 \times C1)$

Calculate C2:

$C2 = C1 \times 1070 + (f0 \times 0.12)$

Calculate C3:

$C3 = 2.0 \times C2$

All calculations for tuning the PLLs on the TONE CONTROL were done assuming C1 equal to 0.1 microfarad. The other timing components were then calculated and, except for R1, the nearest standard values were used. For R1, a 10 to 20 turn potentiometer was used to ensure a fine tuning capability. R1 on each PLL was initially set to its calculated value, then trimmed in a manner described below. Once a value for R1 is determined, a wire wound resistor can be built and used in its place. Tuning the 7 PLLs on the TONE CONTROL is accomplished as described below.

Solder all components in place on the PC board. Place the TONE CONTROL in parallel with your telephone, and connect + 5 volts and ground to the device. Connect the OE input of the octal tristate device to ground. This enables the tristate device so that keypress information is always available on the data bus interface of the TONE CONTROL. Lift the receiver; press the digit 1 on the touch tone keypad and hold it down. The PLLs affected by the digit 1 are the row 1 and column 1 PLLs. Referring to the schematic drawing in Figure 9, and the interface connector in Figure 11, D1, D4, and D7 on the data bus interface will be a logic 0 when the row 1 and column 1 PLLs are tuned properly (D7 is the keypress valid flag and is a logic 0 for all keypresses). With digit 1 depressed, the remaining data bits should be a logic 1 as indicated in Figure 5. If the bit pattern is not as indicated, adjust the trimmer on the row 1 PLL (see Figure 8) until D4 is a logic 0. Now adjust the trimmer on the column 1 PLL until D1 is a logic 0. This sets the row 1 and column 1 PLLs. The rest of the row and column PLLs are adjusted in the same manner. When all the rows and columns have been adjusted, the TONE CONTROL is ready to use. The schematic diagram, provided in Figure 9 should aid in troubleshooting unanticipated problems.

In conclusion, you can connect the TONE CONTROL to your computer, install an appropriate answering cir-

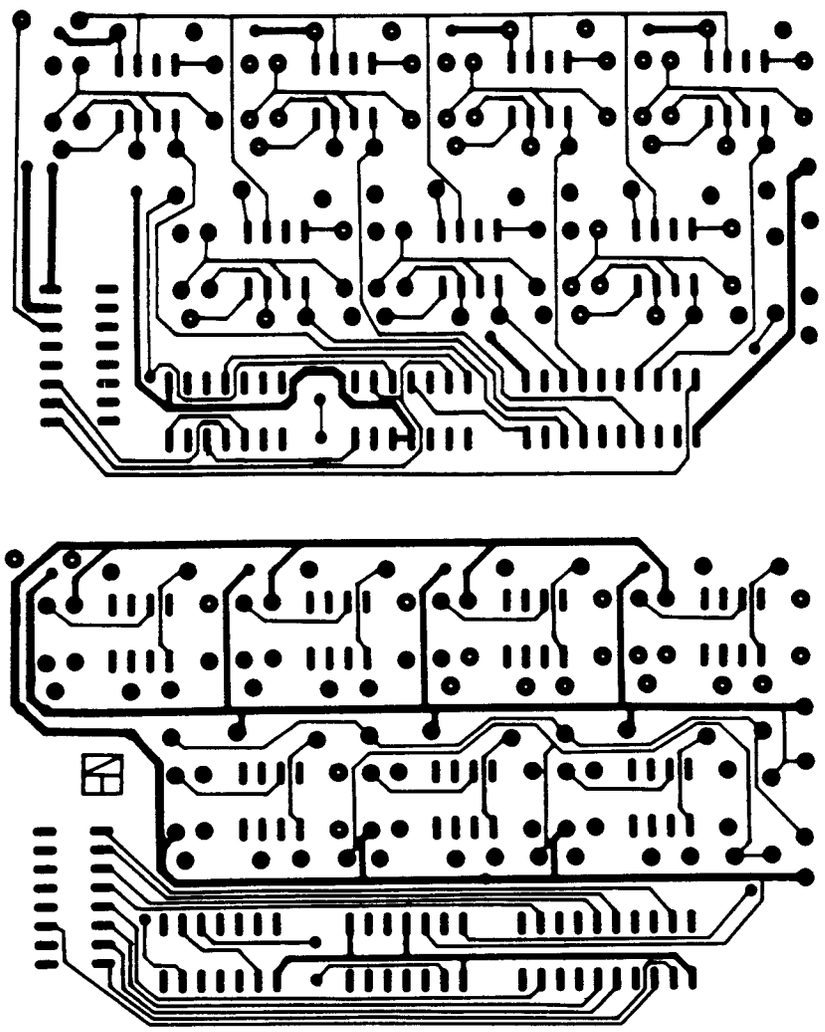
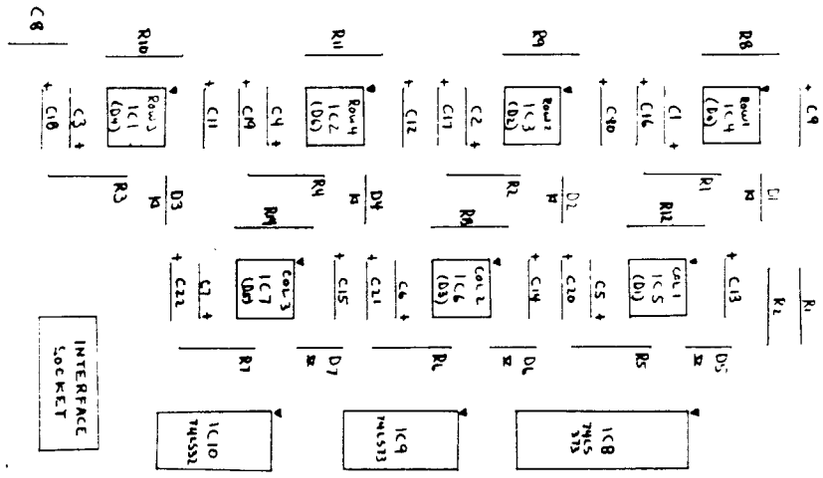


Figure 7: Foil patterns for tone control circuit.

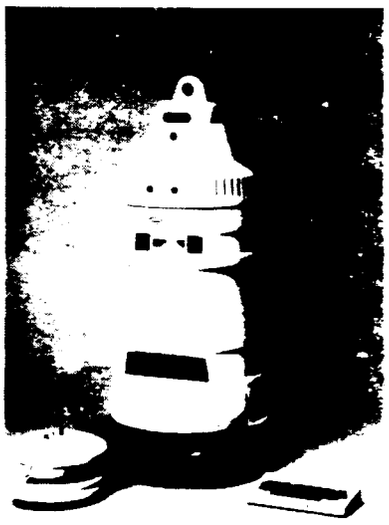


NOTES: ● indicates pin 1
IC4-IC7 are LM567

Figure 8: Parts placement.

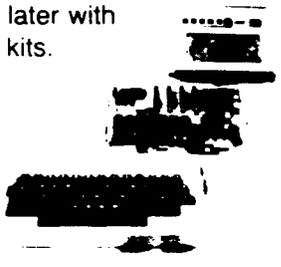
GEMINI
THE AUTONOMOUS ROBOT

IS NOW PRICED FOR EVERYONE!



Buy each subassembly as a kit or factory assembled and create your own GEMINI Robot.

Or, for convenience, start with **GEMINEX** our starter kit, and expand to GEMINI later with upgrade kits.



Either way,
Buy a piece of tomorrow
TODAY!

CALL or WRITE For Our FREE Brochure.

arctec systems
9104 Red Branch Road
Columbia Maryland 21045
(301)730-1237
Telex 87-781

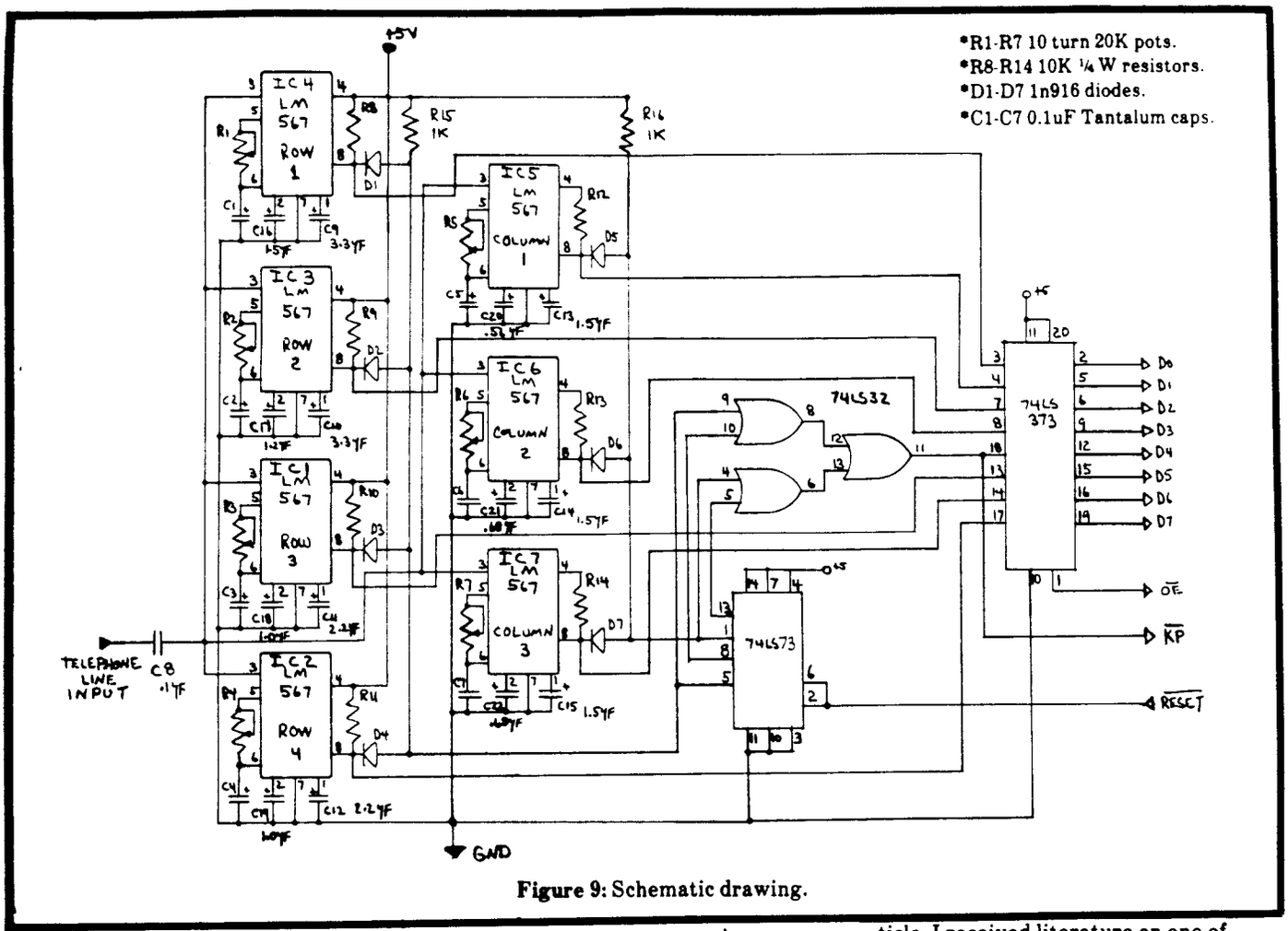


Figure 9: Schematic drawing.

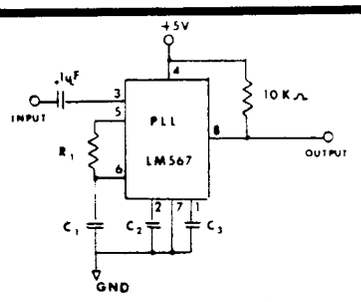


Figure 10: Phase Locked Loop

cuit, and have finger tip control of your computer from any touch tone telephone, without the need for a modem or a terminal. An example of an application where circuits like the TONE CONTROL are being used is in home management systems. These systems control the environmental conditions within the home as well as controlling appliances, security systems, and lighting. In fact, with a little imagination, dozens of applications for telephone control can be created. If you have a unique idea for a telephone control application, drop us a line here at *The Computer Journal*; we are anxious to hear what you have to say.

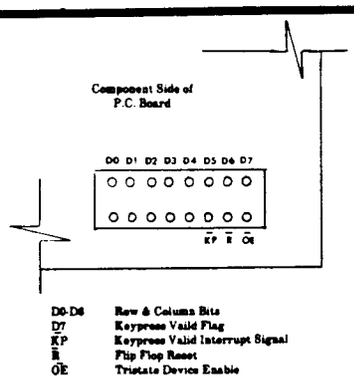
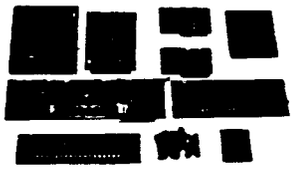

Component Side of P.C. Board
D0 D1 D2 D3 D4 D5 D6 D7
KP R OE
D0-D6 Row & Column Bits
D7 Keypress Valid Flag
KP Keypress Valid Interrupt Signal
R Flip Flop Reset
OE Tristate Device Enable

Figure 11: Interface Connector

Author's Note

When I originally developed the TONE CONTROL circuit a few years ago, there were no single chip devices that could reliably detect the dual tone telephone frequencies. I researched a few devices which claimed to detect the dual tone frequencies, but they turned out to be either unreliable or unavailable, so I was forced to develop the TONE CONTROL circuit. Recently, however, a new generation of dual tone frequency detectors have been made available. In fact, while writing this article, I received literature on one of those devices (the SSI 204), and saw it used in a home management design. I will be obtaining one of these chips soon and will be glad to report my findings to you. ■

MICROCOMPUTERS AND INTERFACES



We have six single board computers, two video boards and interfaces for the IBM-PC and APPLE computers. You can use our products for security systems, heat control, light control, automated slide show, irrigation systems, telephone systems, automated pool control just to name a few. All available. For catalog call or write to:

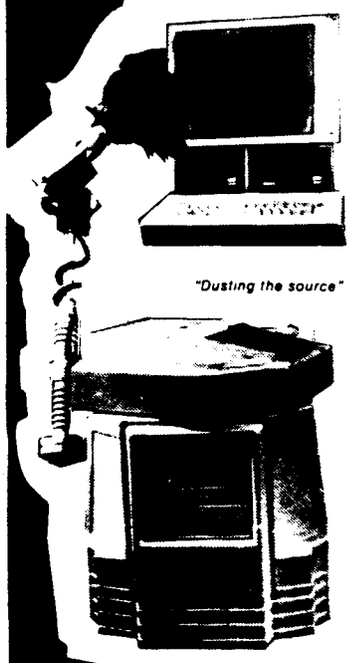
JOHN BELL ENGINEERING, INC.
400 OXFORD WAY
BELMONT, CA 94002
(415) 592-8411

(CBIOS continued from page 15)

```
DPB1:           ;DISK PARAMETER BLOCK FOR DOUBLE DEN., SINGLE SIDED DISKS
                ;       WITH BLOCK SIZE BLKSZ = 2048 BYTES / BLOCK
                ;
        DW      52              ;SECTORS PER TRACK
        DB      4               ;BLOCK SHIFT FACTOR
        DB      15              ;BLOCK MASK
        DB      1               ;EXTENT MASK
        DW      242             ;DISK SIZE-1 (NO. OF BLOCKS/DISK-1)
        DW      127             ;NO. OF DIRECTORY ENTRIES MAX.-1
        DB      192             ;DIRECTORY ALOCATION SPACE MASK, 1 ST BYTE
        DB      0               ;SAME AS ABOVE, 2 ND BYTE
        DW      32              ;CHECK SIZE
        DW      2               ;NO. OF SYSTEM (NOT ACCESSABLE) TRACKS
;
DPB2:           ;DISK PARAMETER BLOCK FOR SINGLE DEN., DOUBLE SIDED DISKS
                ;       WITH BLOCK SIZE BLKSZ = 2048 BYTES / BLOCK
                ;
        DW      52              ;SECTORS PER TRACK
        DB      4               ;BLOCK SHIFT FACTOR
        DB      15              ;BLOCK MASK
        DB      1               ;EXTENT MASK
        DW      242             ;DISK SIZE-1 (NO. OF BLOCKS/DISK-1)
        DW      127             ;NO. OF DIRECTORY ENTRIES MAX.-1
        DB      192             ;DIRECTORY ALOCATION SPACE MASK, 1 ST BYTE
        DB      0               ;SAME AS ABOVE, 2 ND BYTE
        DW      32              ;CHECK SIZE
        DW      2               ;NO. OF SYSTEM (NOT ACCESSABLE) TRACKS
;
DPB3:           ;DISK PARAMETER BLOCK FOR DOUBLE DEN., DOUBLE SIDED DISKS
                ;       WITH BLOCK SIZE BLKSZ = 4096 BYTES / BLOCK
                ;
        DW      104             ;SECTORS PER TRACK
        DB      5               ;BLOCK SHIFT FACTOR
        DB      31              ;BLOCK MASK
        DB      3               ;EXTENT MASK
        DW      242             ;DISK SIZE-1 (NO. OF BLOCKS/DISK-1)
        DW      127             ;NO. OF DIRECTORY ENTRIES MAX.-1
        DB      128             ;DIRECTORY ALOCATION SPACE MASK, 1 ST BYTE
        DB      0               ;SAME AS ABOVE, 2 ND BYTE
        DW      32              ;CHECK SIZE
        DW      2               ;NO. OF SYSTEM (NOT ACCESSABLE) TRACKS
;
DPB4:   DW      60
        DB      5
        DB      31
        DB      1
        DW      2600
        DW      511
        DB      240
        DB      0
        DW      0
        DW      1
;
;       *****************************************
;       *                                       *
;       *               END OF FIXED TABLES     *
;       *                                       *
;       *****************************************
;
STAKSAV:        DS      2       ;STACK SALVAGE LOC
TEMPSTK         EQU     0F7FFH  ;JUST BELOW THE ROM
;
BEGDAT          EQU     $       ;BEGINING OF DATA AREA
DIRBUF:         DS      128     ;DIRECTORY ACCESS BUFFER
;
ALV0:           DS      31      ;ALLOCATION VECTOR 0
CSV0:           DS      32      ;CHECK VECTOR 0
;
ALV1:           DS      31      ;ALLOCATION VECTOR 1
CSV1:           DS      32      ;CHECK VECTOR 1
;
ALV2:           DS      31      ;ALLOCATION VECTOR 2
CSV2:           DS      32      ;CHECK VECTOR 2
;
;
ALV3:           DS      31      ;ALLOCATION VECTOR 3
CSV3:           DS      32      ;CHECK VECTOR 3
;
ALV4:           DS      326
CSV4:           DS      0
BUFFER1:        DS      128
ENDDAT          EQU     $               ;END OF DATA AREA
DATSIZ          EQU     $-BEGDAT        ;SIZE OF DATA AREA
                .DEPHASE
                END     START
```

```
block move 1
switch TPA (bring in new)
block move 2
switch TPA (bring back old)
block move 3
banked bios
.phase F200H (beginning of my bios)
unbanked-bios
.dephase
```

FIGURE 8.



HERO®/APPLE® HANDSHAKE

ROBI. . .an affordable interface for the robotics experimenter. Easy hook-up (8 screws on HERO ®, 1 card slot on Apple ® II or IIe) and a low price are combined with extra capabilities in the ROBI computer/ robot interface.

$199.00!

VISA MasterCard

ROBI SPECIFICATIONS

- *4 programmable bidirectional 8-bit ports for interface and expansion*
- *programmable control over handshaking*
- *access to signals through tie point blocks on robot's Experimental Board*
- *6-foot cable for interface limited remote operation*
- *user-friendly software quickly transfers files between computer and robot, stores and retrieves files to and from disk*
- *not copy protected Software is provided in DOS 3.3*
- *liberally commented source code included*

BERSEARCH
Information Services

26160 Edelweiss Circle
Evergreen, CO 80439
(303) 670-6137

APPLE ' is a trademark of Apple Computer
HERO ' is a trademark of Heath Electronics

New Products

Interface Breadboard

Group Technology has announced their BG-Board Interface Breadboard for the IBM-PC and its compatibles, and for the Commodore 64 and VIC-20 computers. The BG-Board provides safe access to the data, control, and address buses of the microcomputer, allowing the user to construct interfaces for controlling and monitoring home appliances, analytical instruments, temperature control systems, security systems, voice synthesizers, and a variety of real-worl applications.

Individualized interface cards (called CableCards) orient and condition signals to permit the BG-Board to be used with the most popular microcomputers. CableCards are available for the IBM-PC and its compatibles, The Commodore 64, the VIC-20, TRS-80 Models I, II, III, and 4, and the Apple II Oe. Users are not locked into one brand of computer because the board can be used with a wide variety of micros by simply changing an inexpensive interface card.

The BG-Board, which is buffered to protect the microcomputer in the event that wiring errors occur, provides up to eight decoded address outputs in either the device addressing or memory addressing models. Up to 256 input (I/O) devices may be addressed using BASIC software. A built-in logic probe permits logic levels and pulse edges to be detected.

For the IBM-PC, either accumulator I/O or memory-mapped I/O can be used to address devices. Jumper options on the CableCard permit selection of appropiate blocks for addressing devices, accessing the interrupt request, and constructing an interrupt acknowledge.

For the Commodore 64 and VIC-20, only memory-mapped I/O can be used to address devices. More information can be obtained from Group Technology, Ltd., PO Box 87, Route 1 Box 83, Check, VA 24072, phone 703-651-3153.

Stepper Motor Driver

Cyberpak is offering the HS-2, a single card stepper motor controller capable of driving two stepper motors, which interfaces to a personal computer or control circuit. The HS-2 driver interfaces to any 8-bit parallel TTL port, and interfaces to stepper motors (up to 2 amps/phase) used in education, robotics, and machine control applications.

They claim that the HS-2 has performance advantages because of the bipolar chopper drive method used. Bipolar motors can deliver a 30% increase in torque over unipolar motors with the same power, and the chopper mode drivers which regulate current through the coils of the motors yield a much higher maximum step rate than can be achieved by controllers using L/R methods. The HS-2 allows the user to set the current for each motor independently.

The HS-2 board size is 5.0 × 7.4 inches, operates from 12 to 46 VDC, and full or half-step plus a power down mode can be selected via the computer. It is available from stock at $109 for the single motor version or $149 for the dual motor version. For more information contact Cyberpak at PO Box 38, Brookfield, IL 60513 phone (312)387-0802

PROMAL for the Apple and IBM

SMA has announced the availability of **PRO**grammer's **M**icro **A**pplication **L**anguage (PROMAL), a new high-level, structured programming language similar to C and Pascal. It is currently available for the Commodore 64 and 128, the Apple IIe (with 80 column card and PRODOS), and the Apple IIc. They expect it to be available in August 1985 for the IBM-PC, PCjr., PC/XT, PC/AT and all MS-DOS compatible machines.

They state that PROMAL was designed to meet the objectives of simplicity, power, and speed; and that PROMAL's fast one-pas compiler and efficient run-time environment permits applications to be written in a high-level language which previously had to be written in assembler for performance reasons.

It consists of the Executive (operating system) which provides file, memory and program management, and I/O redirection; a full-screen cursor driven Editor, and a Library of machine language subroutines which support the run-time environment with optimized routines for file I/O, string handling, formatted output, cursor control and data conversion. At the source level, the language will be compatible across all target machines.

The Developer's Version which includes an unlimited run-time distribution license is $99.94, and the End-User Version is $49.95. More information can be obtained from Jennifer L. Conn at Systems Management Associates, PO Box 20025, Raleigh, NC 27619 phone (919) 787-7703.

Laboratory Data Acquisition Catalog for Apple II

Interactive Microware has available a new 64 page catalog describing its line of data acquisition hardware and analysis software for Apple II microcomputers.

Included are the general purpose ADALAB® data acquisition interface card and hardware accessories; data acquisition software; application software for gas, liquid, and gel permeation chromatography, temperature monitoring, and also for mulit-channel, multi-purpose data logging and process control; software for graphics plotting, drafting, and design.

Contact Kay Whiteside at Interactive Microware, PO Box 139, State College, PA 16804-0139 phone (814) 238-8294.

C Screen Management Utility

CompuCraft has announced cVIEW 2.11, a screen management tool for software developers writing in C. cVIEW allows the developer to modify his forms without having to change the code behind them, in fields are defined as they are placed on the screen, type specification of the input fields provides automatic testing and rejection of incorrect user entries, minimum/maximum range limits may be specified for numeric fields, user written edit routines may be applied to any field, and all of the special keys can be defined by the programmer for each form.

cVIEW runs on the IBM-PC or compatible, and can be used with the Computer Innovations, Mark Williams, Microsoft or Lattice C compilers. The cVIEW screen package which contains the editor used to create and modify forms, online help for each level of operation, the runtime library required for interfacing to applications programs, and programmers reference manual is $245, and a demo disk is available for $25 which is credited to the purchase of cVIEW. Call or write Jerry Januzzi at CompuCraft Corp., 42101 Mound Road, Sterling Heights, MI 48078 phone (313) 731-2780.

Hero Robot Macros

Bersearch Information Services has developed a software package for programming the Heath-Zenith Hero-1 Robot with an Apple II. The package allows the user to program with easily remembered mnemonics and base-10 numbers in a BASIC format. The Robi interface is used to transfer finished programs from the Apple to Hero.

Tom Bernard, at Bersearch Information, has written Hero Macros for the S-C Software 6800 Cross Assembler. The Hero Macros allow the user to program Heath's Robot Interpreter Language with easily remembered mnemonics, for example the line **1130 >MVWRIM GRIP,OPEN,60,FAST** instructs Hero to open his gripper 60 units at fast speed. As in BASIC, line numbers are used to enter lines, and the cross assembler includes auto line numbering to make programming easier. The programmer uses labels to target branch points and data addresses; the assembler calculates the actual addresses from the labels.

The Hero Macros come with 30 pages of documentation, and the Cross Assembler and Robi Interface are also well documented. Maximum motor positions (in base-10 and hex.) are given, along with each command's exact syntax. The Hero Macros disk includes ready to run sample programs to illustrate programming technique. Customer support telephone numbers are included.

The Cross Assembler with Hero Macros sells for $100.00; the Robi Interface sells for $199.00. Both as a package sell for $279.00. To order, or for more information, contact Tom Bernard at Bersearch Information Services, 26160 Edelweiss Circle, Evergreen, CO 80439 phone (303) 670-6137.

Classified

The Computer Journal will carry Classified Ads. The rate is $.25 per word. All Classified Ads must be paid in advance, and will be published in the next available issue. No checking copies or proofs are supplied, so please type your ad or print legibly.

KEYBOARDS FOR COMPUTER BUILDERS. Full ASCII, numeric pad, UC/lc, CAPS-LOCK, REPEAT, SELF-TEST! Brand new, hundreds sold to builders of Apples, Xerox 820s, Big Boards, etc. Parallel TTL output, strobe. 5 volts/100 ma. Custom case available. Keyboard $35. Documentation (21 pgs.)/cable pkg. $5. Spare CPU/ROM $4. UPS included. Detailed specs on request. Electrovalue Industrial Inc., Box 376-CJ, Morris Plains, NJ 07950. (201)-267-1117.

Voice Processor for the KAYPRO Computer. Unlimited speech contains all software. Call or write Busch Computer, PO Box 412, West Haven, CT 06516. Phone (203)484-0320.

S-100 68008 CPU BOARD. Detailed description in issue 16 of The Computer Journal. A&T $260, Kit $210, Bare Board $65. Prices include shipping. INTELLICOMP, INC., 292 Lambourne Ave., Worthington, OH 43085. Phone (614)846-0216 after 6 p.m.

S-100 Bus IEEE-488 Interface Card with cable, manuals, and software for North Star Horizon. Purchased new From Pickles and Trout in 1979 and used once. $100. Call Phil Wells at (406) 755-1323 days or (406) 257-5326 evenings.

Morrow Decision I S-100 system with MPZ-80 CPU, DJ/DMA floppy disk controller, 256K static ram, Wonderbus I/O on mother board, Disk Jockey Hard Disk (HDCA) Controller, 801 floppy drive, 10MB hard disk, CP/M, Micronix Multiuser system, unconfigured MP/MII, dBase II, Wordstar, Accounting Plus. Excellent condition. $3500, some trades considered. TCJ, 190 Sullivan Crossroad, Columbia Falls, MT 59912 Phone (406)257-9119.

Book Sale—These books are offered at this price while the supply lasts.

Zilog Z80-CPU Technical Manual, $1.50

The Programmer's CP/M Handbook
by Andy Johnson-Laird, $18.66

Real Time Programming—Neglected Topics
by Caxton C. Foster, $8.46

CBASIC Users Guide
by Osborne, Eubanks, and McNiff, $15.26

Introduction to FORTH
by Ken Knecht, $9.31

FORTH Programming
by Leo J. Scanlon, $14.41

Interfacing and Scientific Data Communications and Experiments
by Peter R. Rony, $6.76.

These prices are postpaid in the U.S. only. TCJ, 190 Sullivan Crossroad, Columbia Falls, MT 59912.

Corvus 10MB Hard Disk for the Apple II plus, $888.00; Apple III Second Disk Drive, $199.00; Apple III SOS Device Driver Writer's Guide $19.95; BPI Accounting for Apple III (Requires Hard Disk) $99.00; Apple Writer 1.1, 16 Sector, $8.88; Apple DOS User's Manual (II, II plus, IIe), $8.88; Apple DOS Programmer's Manual (II, II plus, IIe), $6.88; KAYPRO-Home Accountant by Continental, $49.00; Soroc IQ 130 Terminal, $399.00. All plus shipping. The Computer Place, 36 2nd Street East, Kalispell, MT 59901, Phone (406)755-1323.

The Security Disk. Protected vs unprotected. At last, the best of both worlds. Here is software designed to protect your private files, plus allow you to analyze and unlock your other other "Copy-Protected" disks, then change them to standard DOS 3.3 format. Simple "Password" protection to "Cryptology". A disk packed with secrets, tips, and other goodies. Not locked-up. Completely REMarked. Machine Source Codes included. Supports Apple II, II Plus, IIe, and IIc. To order send $24.95 CHECK/MO to B.M.E. Enterprises, Box 191-J, Kila, MT 59920.

Multitasking and Windows With CP/M-80

A Review of MTBASIC

by Art Carlson

Have you ever wished that your computer could do two or three things at the same time? We often feel this need when programming measurement or control applications, and while large systems usually have this ability (it's called multitasking) it has not been readily available on eight bit CP/M systems. When I saw Softaid's ad for MTBASIC which theyclaimed could do multitasking and windows under CP/M-80 or PC-DOS, I contacted them for a review copy. I received an 8" SSSD CP/M disk to run on my Morrow S-100 system with a Z-80 CPU and their 99 page 8½ × 11 manual.

MTBASIC is similar to other BASIC's with many familiar commands, but Softaid has made no attempt to make it compatible with the others because MTBASIC has many special features. MTBASIC is an interactive compiler which functions like an interpreter. The program is entered with line numbers similar to other basics, but when you enter the direct command RUN, it compiles the program into object code and then executes the program. After debugging, the compiled code can be written to a disk file for execution as a standard stand-alone .COM file, and there is no license fee for the object code. You just have to credit Softaid in the documentation of any compiled programs sold or given to others. All .COM files produced by DISK COMPILE are ROMable.

My primary interest in MTBASIC is the ability to run several activities concurrently in measurement and control applications, and while Softaid states that the multitasking feature adds a degree of realism to games I'll leave that feature for others to investigate. We realize, of course, that a single microprocessor can not really run more than one task at a time, but with concurrent processing the compiler switches the processor from one task to another so rapidly that it appears that they are all running at the same time.

A secondary feature of MTBASIC is the ability to provide pop up windows under program control while working with a character oriented terminal (I'm using a TeleVideo model 950). MTBASIC can be run as received if you do not use windowing, but you may have to use the supplied INSTALL.BAS program to configure your terminal to support windowing. The CP/M version is supplied pre-installed for the ADM-3A (identical to Kaypro) and ran on my TeleVideo 950 with no alteration.

Interrupts for Multitasking

MTBASIC can use either software or hardware interrupts to schedule the tasks for multitasking. The command TICS ON before the program is compiled will generate software tics in the program, and this is the easiest method to use. Hardware interrupts are more difficult to use because the programmer must write an interrupt service routine to process the hardware interrupts. Software interrupts come at a rate proportional to the amount of time it takes to execute each individual statement within the program, which varies considerably. Therefore precise timing is not possible with software interrupts. Hardware interrupts, while more complex because you must write the service routine, can be designed to generate a very precise frequency and to run a task at a specific time.

MTBASIC also supports device interrupts, which are useful because the program does not have to continuously poll a particular device to determine if data is ready. A disk service routine could be written in MTBASIC by dedicating a task to servicing the device interrupt.

A multitasking program consists of two or more tasks which run asynchronously with respect to each other. Unless the programmer uses semaphores to provide some sort of synchronization of execution, it is difficult to tell when any one task will be executing. Tasks are not like subroutines. A subroutine only runs when it is called, and terminates when its return instruction is executed. A task, on the other hand, may be running at any time, as computer time is shared between execution of the tasks. A particular task generally does not run to completion before the computer starts running another task. The computer just suspends execution of one task and goes to another. Eventually it picks up with the suspended task where it left off and resumes execution. Although each task is executed in a "choppy" fashion, to the user it appears as if all tasks are executing smoothly because the computer is so fast.

Scheduling Tasks

The main MTBASIC program is known as the *Lead Task* and can consist of the entire program (you won't always be using multi-tasking). The lead program can start one or more other tasks, and these tasks can start tasks. The tasks are started with the RUN statement (not to be confused with the direct command of the same name) which includes the task number and the schedule interval as its arguments. If the specified task is terminated with the EXIT statement, it is automatically restarted after the number of tics specified in the schedule interval. The schedule interval must be in the range of 1 to 32,767 tics, but the number can be increased to any value by setting a flag which counts the number of times the task has been executed and allows the task to continue executing only if a certain number of counts have been detected. The program listing in Figure 1 is a simple example showing the use of the RUN and Task statements.

Windowing

MTBASIC allows you to provide windows with a few simple commands, but you can ignore this feature and program in a normal nun-windowing mode if you do not need windows. A window is a subsection of the CRT screen and any window can be any size up to full screen. Whenever a program selects a window and sends output to it, all output will go to that window. A window's borders are barriers to the PRINT and FPRINT statements, and inhibit these statements from writing

anywhere but within the currently selected window.

Programming With MTBASIC

The fact that a language can provide multi-tasking and windows is only part of the story. Equally important is how well the language handles the rest of the programming tasks. I feel comfortable with MTBASIC, and appreciate the fact that you can pick up the phone and talk to someone who understands your questions.

MTBASIC source programs are entered with line numbers, and each line is checked for errors when you hit the return at the end of the line. And there are meaningful error messages (28 of them), no more SYNTAX ERROR, but rather something like QUOTE OR PARENTHSIS MISMATCH, or STATEMENT FORMED POORLY. There are also run-time messages such as UNMATCHED FOR...NEXT PAIR and ILLEGAL PRINT FORMAT. These error messages are fully explained in the manual.

After typing the source program, you enter the direct command RUN, and the code is compiled and then run. And the compiler is fast! A sample 145 line program was compiled and running less than two seconds after I hit the return. No more exiting to the system, calling a separate compiler, and then using a LOAD program. Just enter RUN, hit the return, and the program is compiled and running before you can reach for your cup of coffee. You can save your source code to disk, and you can compile to disk (which results in a stand-alone .COM file). You will also have to compile to disk if the program is too large for both the source and the compiled code to fit in memory at the same time. In fact, the DISK COMPILE command has been modified to allow very large source files to be compiled to disk from a source file on the disk.

The LOAD command brings a file in-

Direct Commands

BYE	Exits MTBASIC and returns to CP/M
COMPILE	Compiles a program without running it
CONSOLE	Directs output to console (disables PRINTER)
DISK COMPILE	Compiles a program to a .COM file
END	Marks the end of a source program file
ERROR	Turns on runtime error checking (default is ON)
GO	Starts an already compiled program running
LIST	Displays the program code
LOAD	Reads a source file from disk
NEW	Erases the current program
NOERR	Turns off runtime error checking (default is ON)
PRINTER	Sends output to the printer (disables CONSOLE)
RUN	Compiles and runs a program
SAVE	Saves a program's source code to disk
TICS	Turns software interrupts on or off (default is OFF)
VARIABLE	Sets RAM addresses (CP/M only)

Figure 2:

FREE SOFTWARE

RENT FROM THE PUBLIC DOMAIN!

User Group Software isn't copyrighted, so there are no fees to pay! 1000's of CP/M and IBM software programs in .COM and source code to copy yourself! Games, business, utilities! All FREE!

CP/M USERS GROUP LIBRARY
Volumes 1-92, 46 disks rental—$45

SIG/M USERS GROUP LIBRARY
Volumes 1-90, 46 disks rental—$45
Volumes 91-176, 44 disks rental—$50
SPECIAL! Rent all SIG/M volumes for $90

K.U.G. (Charlottesville) 25 Volumes—$25

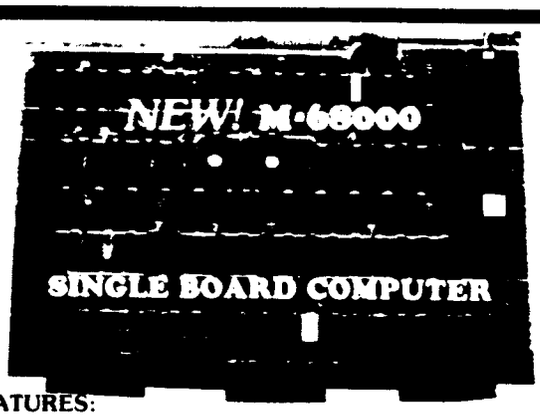
IBM PC-SIG (PC-DOS) LIBRARY
Volumes 1-200, 5¼" disks $200

174 FORMATS AVAILABLE! SPECIFY.

Public Domain User Group Catalog Disk $5 pp. (CP/M only) (payment in advance, please). Rental is for 7 days after receipt, 3 days grace to return. Use credit card, no disk deposit. Shipping, handling & insurance—$7.50 per library.

(619) 914-0925 information,(9-5)
(619) 727-1015 anytime order machine
Have your credit card ready! VISA, MasterCard, Am. Exp.

Public Domain Software Center
1533 Avohill Dr.
Vista, CA 92083



FEATURES:

16 bit Motorola 68000 CPU operating at 5 MHz or 10 MHz, 20K of on board fast static RAM, 16K bytes of on board EPROM space, 7 autovectored interrupts, 3 memory/device expansion buses, 2 serial communication ports (RS-232 C), 16 bit bidirectional parallel port, 5-16 bit counter/timers with vectored interrupt and time of the day clock. On board monitor allows to download and debug programs generated on APPLE II, TRS-80 and CP/M using our M68000 Cross Assembler.

PRICE:

M68K Bare board with documentation	$ 99.95
M68MON monitor & mapping PROM's	$135.00
MD512 Memory/Disk Contr. (Bare Board)	$ 99.95
M68KE Enclosure with P.S. and card cage	$249.95
M68000 Cross Assembler	$149.00
M68K Documentation only	$ 15.00
Shipping & handling (Domestic)	$ 4.50
(foreign)	$ 20.00

CALIFORNIA RESIDENTS ADD 6% TAX

EMS Educational Microcomputer Systems

P.O. BOX 16115, IRVINE, CA 92713
(714) 854-8545

```
10 INTEGER A:   REM DECLARE INTEGER VARIABLE
20 REAL X:   REM DECLARE REAL VARIABLE
30 A = 0:X = 0:   REM INITIALIZE VARIABLES
40 RUN 1,100:   REM RUN TASK 1 EVERY 100 TICS
50 RUN 2,500:   REM RUN TASK 2 EVERY 500 TICS
60 GOTO 60:   REM WAIT HERE TILL TIME FOR TASK
70 TASK 1:   REM CODE FOR TASK 1 FOLLOWS
100 TASK 2:   REM CODE FOR TASK 2 FOLLOWS
END
```

Figure 1:

Statements

CALL	Starts an assembly language subroutine
CANCEL	Stops a task
CLOSE	Closes a file
CURSOR	Positions the cursor in a window
DATA	Defines a group of constants
DELETE	deletes a file
ERASE	Clears the entire CRT
EXIT	Terminates a tesk
FILE	Selects an I/O device
FOR/NEXT	Loop control
FPRINT	Fromatted print
GOSUB	Subroutine call
GO TO	Program branch
IF	Decision
INPUT	Enter data from i/O device
INPUT$	Enter data, including commas
INTEGER	Defines integer variables
INTMODE	Defines interrupt mode (CP/M only)
INTON	Turn interrupts on
INTOFF	Turns interrupts off
JVECTOR	Defines interrupt vector (CP/M only)
OPEN	Opens a file
OUT	Output to an I/O port
POKE	Modifies a memory location
PRINT	Outputs data
RANDOMIZE	Seeds the random number generator
READ	Gets data from a DATA statement
REAL	Defines floating point variables
REM	Comment, also denoted by "!"
RESTORE	Selects a DATA statement
RETURN	Return from a subroutine
RUN	Starts a task going
SEEK	Random file I/O record position
STRING	Defines string variables
TASK	Defines the start of a task
TRACE ON	Prints line numbers as they are executed
TRACE OFF	Disables TRACE ON
VECTOR	Links to interrupt vector
WAIT	Delays a task's execution
WCLEAR	Erases a window
WFRAME	Draws an outline around a window
WINDOW	Defines a window
WSAVE	Saves the contents of a window
WSELECT	Selects a window
WUPDATE	restores a saved window
:	Seperates multiple statements on a line

Figure 3:

to the compiler from disk, and the program is checked for syntax errors as the program is read. The files are standard ASCII, and this is fortunate, because MTBASIC does not inculde an editor. Programs can be created offline with your favorite text editor (use the non-document mode with WordStar), and then checked for errors as it is loaded. LOAD does not erase programs already in memory, and you can merge two or more programs which will be mixed as a function of the line numbers. This is a great way to incorporate standard routines as long as you are careful about the line numbers, and I understand that a renumbering utility is included on the current disks. To avoid unwanted merging of programs—believe me it produces some startling results when you don't expect it—just enter the command NEW before LOADing the next program.

Variables must be declared (INTEGER, REAL, or STRING) at the beginning of the program before any executable code is encountered, and should be initialized. Earlier versions did not support string arrays, but this was added in January of this year. INTEGER variables are stored using a sixteen bit two's complement representation and can range from + 32,767 to – 32,767. Positive values which exceed 32,767 will appear as negative numbers. Real values are four byte (32 bit) IEEE compatible single precision real numbers. MTBASIC does not support BCD or 64 bit double precision numbers. Constants with a decimal point are assumed to be real numbers, and hexadecimal constants can be used with a leading dollar sign. The compiler provides automatic mixed mode expression evaluation, but you should be aware of the possible loss of precision because all components are converted to real numbers if any component is a real number.

MTBASIC is a completely recursive language, that is, a routine can call itself. Common applications for recursion are mathematical functions such as computation of factorials, processing linked lists, and binary trees. There is a limit on how deeply the program can call itself which depends on the nature of the programs. They recommend that you limit recursive programs to a depth of ten to prevent the stack from overflowing, although programs with a depth of fifty have run successfully. Recur-

sive programs should not be combined with multitasking programs, since both recursion and multitasking use large amounts of the stack.

Listings of the the Direct Commands, Statements, and Functions are shown in Figures 2, 3, and 4, and there are several interesting features which I have not had time to try. The arguments for the trig functions are in degrees, which I prefer to use for real world applications, instead of in radians as with AppleSoft and MBASIC. The CALL statement to begin execution of a machine language subroutine has been improved so that arguments may be passed to the routine in the form of the address and mode of the variable. The ADR function returns the address of a variable so that arguments can be passed to assembly language subroutines, or an assembly language routine may be POKEd into an array whose address has been determined. A number of new statements and functions have been added to perform random I/O in a fashion very similar to that used by Microsoft's MBASIC. The random files are written in binary, so a text editor cannot be used to examine them.

Functions

ACOS	Arccosine
ADR	Returns variable address
ASC	Returns ASCII value
ASIN	Arcsine
ATAN	Arctangent
BAND	Bitwise AND
BOR	Bitwise OR
BXOR	Bitwise exclusive OR
CHR$	Returns string equivalent
CONCAT$	Concatenates two strings
COS	Cosine
ERR	Returns error numbers
ERR$	Returns error messages
GET	Returns one character from the surrent file
INP	Reads input expression from input port
KEY	Returns one ASCII value from console
LEN	Returns the length of a string
LOG	Natural log (base e)
MID$	Returns part of a string
PEEK	Returns the contents of a memory address
RND	Generates random numbers
SIN	Sine
SQR	Square root
STR$	Converts numbers to strings (converse of VAL)
TAN	Tangent
VAL	Converts string to numbers (converse of SRT$)

Figure 4:

Conclusions

MTBASIC would be a bargain at $49.95 even if the compiler was not included. It's worth the price just to try the multitasking and windows even if you don't intend to use them, but I'm sure that you will use them once you've tried them. Softaid is improving and updating the program frequently, and updates are only $20 plus the original disk! There is no run-time fee for the compiled programs you produce (just give them a credit line).They also publish a user's group newsletter with handy tips and information.

I have only touched on a few of MTBASIC's features in this review, and have talked about the CP/M version. There are a few minor differences in the PC-DOS version, and I understand that a version for the IBM-PC with 8087 support is now available for $79.95 (upgrades are $30 plus your original disk).

We should encourage smaller companies who provide good low-priced software with frequent updates at a reasonable price plus friendly customer support. Contact Cathy Ganssle at Softaid, Inc., PO Box 2412, Columbia, MD 21045-1412 phone (301) 792-8096. And after you get MTBASIC, send your tips, applications, and program listings to *The Computer Journal* so that we can share the information with others. ■

Upcoming Articles

The following list is a sample of some of the interesting articles which are in process. Your suggestions for future articles are welcome.

- Source code drivers for the NEC 7220 graphics chip.
- Accessing the Apple IIs graphics from within a CP/M program using a Z-80 card.
- Kit Building – soldering, desoldering, and repairing printed circuit boards.
- An S-100 WS2797 floppy disk controller board for CP/M 68K.
- Programming the 6522 VIA.
- Stepper Motors – Theory, programming, applications.
- The IEEE-488 Interface – Tutorial and applications.
- Robotics Control applications.
- Programming the Z-80 CTC, PIO, and SIO chips.

Advertiser's Index

Apropos 12
Artec 21
Barnes Research 18
John Bell 22
Bersearch 23
Business Utility Software 6
BV Engineering 31
Classifieds 25
Computer Trader 12
EMS 27
KC Systems 12
Miller Microcomputer Services 3
Next Generation Systems 5
Public Domain Software 27
Rio Grande Robotics 12

Reviewers Needed

We are looking for qualified people to review technical programs and hardware for The Computer Journal. We do not need reviews of Lotus 1-2-3 or similar spreadsheets, wordprocessors, or general business type programs; we'll leave that to the general interest magazines. What we do need are reviews of compilers, assemblers, disassemblers, debuggers, programming utility libraries, scientific and engineering programs, data acquisition and analysis programs, operating system enhancements, and similar items which are used by programmers.

We are also interested in reviews of specialized hardware such as A/D and D/A interfaces, EPROM programmers, stepper motor controllers, and kits—but not most new computers or peripherals, unless there is some technical aspect of special interest to our readers.

We prefer reviews from people who are actually using the product rather than from someone who reviews many different products without using any one of them long enough to become completely familiar with all of its features. The reviews should be truthful and should tell it like it is, but the best reviews are the ones you write about products that you like and want to encourage others to use.

If you are interested in writing reviews, send us a short letter with your background and qualifications, and a phone number where you can be reached in the evening. Include products which you now have available for review, and also items which you would be interested in reviewing if we could obtain a review copy. ■

(Editor continued from page 1)

Knowledge is for sharing. Are you willing to contribute software or information for the exchange? If so, contact me and we'll get it started!

The Only Thing Constant is Change

After years of explosive growth, there are signs that the microcomputer market is fragmenting into a number of areas each with its own requirements. At first almost everyone was technically oriented and either built their machine or was personally involved with keeping it up and running. Then, as business applications were developed, the users were more interested in what the micro could do for them rather than how it worked. At this time the systems still required tender loving technical care by someone who knew the machines, but basically the same hardware was used by hackers, hobbyists, and business. There was one broad market for producers to aim at. Now, there are separate markets developing for business, industrial, serious computerists, and home computer applications; and while there is some overlap, the same hardware will not meet the requirements for the different markets.

The business market is the largest in both dollars and the number of units, and except for a few specialized vertical applications, the large amount of money needed for promotion and large scale production limits it to big business. Competing in this field is like trying to sell cars against General Motors.

The home computer market is still a blood bath, and will remain so until a standard is developed which allows the consumer to freely mix software and components from different vendors, and someone develops the Visa-Calc type blockbuster which makes the computer useful and desireable in the home. Games, checkbook, and recipe programs won't do it!

The industrial and real world applications market is showing signs of very strong growth and is getting ready to explode, but the current work is in developing very specialized applications and this is the least visible of the markets. There is a need for sophisticated software and hardware people who are willing to learn how to apply their skills to measurement, numeric machining, process control, and the control of electro-mechanical devices. These techniques will filter down to consumer product applications (for appliance type products containing a microprocessor, NOT computers as such), resulting in a huge demand for people familiar with programming and debugging utilities and constructing hardware prototypes. The programs, which will be in ROM, will be small (but difficult because of the size) and the successful people will be experienced "roll up the sleeves" "make every byte count" practical minded individuals who know both programming and hardware. I can't think of a better description of a serious hacker.

The remaining market is the serious computerists (who we called hackers before the popular press redefined the word). These people are active in all of the computer areas, and are interested in the computers themselves—not just what they can do, but also how they do it. While this market is small compared with the business and industrial markets, the serious computerists are the ones working on the new developments which will be the basis of the future products for the other markets. This market is showing signs of renewed life, with increased interest in software utilities such as debuggers, assemblers, disassemblers, and emulators; and in hardware for single board computers, EPROM programers and burners, dedicated microcontrollers, A/D and D/A boards, and kits.

We feel that *The Computer Journal* can be most useful in supplying information for the serious computerists, those who want to learn more about their computer, and those interested in real world and industrial applications.

This is YOUR Magazine

We have a lot of good solid technical articles in progress, but we need input from YOU! We will be sending a survey form, but until then, take the time to write and let us know what changes you would like to see in the magazine. ■

(Computer Corner continued from page 32)

sided, 40 track). These are two thirds height and white, but do not let the color fool you – they are all but pure. Lots of these units work at first for a few hours, but then stop ever more. I got four of them and wish I had none, but alas there may be hope yet. I have put considerable hours into finding solutions for their ills. So far I seem to be getting somewhere and would like input from others too.

There seem to be several problems, from poor design to bad parts. The physical slides for the heads can be noisy and sticky, so some lubrication may be needed here. A quick cleaning first may be needed and then light oil or grease will help. The alignment is generally OK, and moving the guides has little effect, however the azimuth may be off on higher tracks which is caused by the guides. Moving and rotating the upper rail (the end has no support) may correct some azimuth problems. If speed of the spindle becomes excessive (starts sounding like a jet airplane) two problems are possible. First the chip, a MJE210, could be bad and need replacement, but secondly and maybe more common is the mica insulator under the chip. If the screw is tightened too far when mounting the chip, the mica can be cut causing the device to be shorted out to case ground. Backing off on this screw will generally help. Also one unit had a loose wire going into the motor and recrimping it helped for sure.

On the PC board there appears to be a shortage of bypass caps especially where the power enters the board. The original schematic had the choke after the bypass caps, but the actual layout has the caps after the coil. Since several parts of the board are fed from before the coil, it could be considered as unfiltered feeds and susceptible to noise. I have added caps at some places I considered important and it seemed to help, but was not a complete solution. My problems and errors did decrease considerably but not completely, so the caps may help and you should use your discretion here. What got me checking that problem was the excessive current drawn and the corresponding spike. I tried this unit on an older supply and found it would not run the machine. On more checking it was decided that a supply of at least 2 amps at 12 volts is needed, less current ability may add to the problems. The book however puts peak current at 1.6 amps which I feel is not correct.

Several changes were made between board designs and some bypasses were removed from the read amp circuit. I am not sure yet if this effects the read, but I have determined that reads are my most common problem. A disk that has been causing problems, was realigned and then written to. This formatting showed many errors, and when checked with findbad would also show errors. However there were no errors when tested on a good drive, showing that the write was OK but the read after write was not. I adjusted R24 on two units and 98% of the errors went away. The other unit went from 200 plus bad sectors to under 30, but this unit doesn't have any bypasses added. R24 is the balance control for the opamps, and tweaking for minimum errors works just fine (the point of no errors is rather sharp). I will try adding the old bypasses to this circuit and see if it helps matters any.

This by no means is a complete solution to all problems but should help you work on them. There are some other people also working on the problem and I would like to hear from them myself. Through correspondence with magazines like f2The Computer Journal we can make happy users out of upset buyers.

Lastly is what happened on May 4, when our club threw an affair and nobody came. We had a great time, and the speakers were terrific, but only sixty people showed up. I fear this was not so much a lack of our abilities to tell people about the event as an indication of changes in computer interest. Had our speakers been in IBM blue suits and talked more on software than on how it was ten years ago in starting a computer company, I guess more might have come. Bill Godbout and George Morrow had a lot to say about problems down the road if we allow governments to use computers in the wrong ways. David Thompson of Micro Cornucopia gave us some insights into why Microsystems went under (Ziff-Davis needed 75,000 subscribers on record for accounting purposes, so combined mailing list kept him from paying advertisers back). Our locals added some color to the affair, but all was dampened by the low show. If you are having an event this year, keep us in touch on how it turns out, for I would like to see if a new trend is forming. This is especially important for clubs as they need to serve their members needs, and usually members just stop coming instead of trying to change their clubs. We now have plans on going after the IBM kit builders, as well as those who want to know more than how to turn it on. Hopefully next month we will have more to say about my other projects that have been put aside because of troubled disk drives. ■

FREE CATALOG AND SIGNAL PROCESSING BOOKLET

CP/M MSDOS AFFORDABLE ENGINEERING SOFTWARE TRSD PCD

CIRCUIT ANALYSIS
- Fast Machine Code
- Complete Circuit Editor
- Free Format Input
- Worst Case/Sensitivities
- Full Error Trapping

ACNAP Version 2.0 $69.95
- Any Size Circuit
- Input / Output Impedances
- Monte Carlo Analysis
- Transients (with SPP)

DCNAP $59.95
- Compatible Data Files
- Calculates Component Power
- 30 Nodes / 200 Components

SIGNAL PROCESSING

SPP $59.95
- Linear/Non-Linear Analysis
- FFT/Inverse FFT
- La Place Transforms
- Transient Analysis
- Time Domain Manipulation
- Spectra Manipulation
- Transfer Function Manipulation
- Editing and Error Trapping
- Free Format Input
- ASCII and Binary Files
- Fast Machine Code

VISA ● MASTERCARD

GRAPH PRINTING
- Linear/Logarithmic
- Multiple Plots
- Full Plot Labeling
- Auto/Forced Scaling
- Two Y-Axes
- ACNAP/SPP Compatible

PLOT PRO $49.95
- Any Printer
- Vertical/Horizontal

PC PLOT $59.95
- Screen Graphics
- Pixel Resolution
- Epson Printer

BV Engineering
Professional Software 2200 Business Way, Suite 207 • Riverside CA 92501 • USA (714) 781-0252

THE COMPUTER CORNER

A Column by Bill Kibler

Well here goes another month of what's been happening. I suppose if nothing was going on there wouldn't be any need for this column. Actually that is what a lot of magazines would like you to think about the non IBM machines, but reality is quite different. My last month has been very busy and I am rushing this out so I can get on to other pressing problems.

An important point has been made by one of our readers and we would like to thank him for both writing and reading our work. The response dealt with my view of the future and my not so great like for IBM's. I agree with him but found a recurring statement that troubles me. In reading my unsolicited copy of PC TECH JOURNAL (I was a Microsystems subscriber), I kept finding the term "standard" used everywhere in referring to the IBM PC. When the letter also used standard several times, I felt obligated to discuss standards as they apply to computers.

In the computer industry there are two types of standards, official and de facto. The official standards are derived by a standards committee which is formed by the IEEE and given the task of formulating the specification under which the standard will operate ever after. Now these standards refer to both hardware and software protocols and usually come after a product has been manufactured and used for several years. The process will take 3 to 5 years before a final agreement can be reached. Some current standards are the RS-232 serial interface, Multibus I and II, IEEE 696 or S-100 bus, and RS 422. There are so many others that it is impossible to cover them all in this column. Once formulated these standards can provide about 90% compatibility between different vendors products.

The de facto standards are ones in which a product has gained enough popularity for a large number of vendors to produce products around it. The Apple II's and IBM PC's are two examples which have many compatible vendors. These so called compatible products are not standards in the true sense, as one manufacture can change them to suit their own needs at any time. For vendors who produce products based on these designs, any changes by the originator means changes for them too. With real standards, there can not be any changes without the whole group agreeing and then the standard usually will be in everybody's best interest. Another important consideration is the need for a long product life, as it may take ten years just to become an accepted standard, and for the product to become bug free.

If you have followed my logic so far you can see that the IBM PC is in fact not a standard product of any kind but a de facto product with enough followers to make it appear as a standard to the casual observer. In reality the product even falls short of any standard. Physically the bus used on the expansion slots has already been upgraded in the AT models and many of the earlier cards will not work. A good physical standard has upward compatibility paths, as well as room for features which are present in the new design but not in the old. For software compatibility the IBM PC is quite lacking in accepted format as entry into the BIOS is to addresses well within the copyrighted PROM and not to fixed tables that can easily be duplicated. Any standard is public, and can be duplicated without threats of law suits. CP/M BIOS and BDOS entry points are fixed tables at the beginning of the program and have been added to but not rearranged since the original version. This allows all past programs to work, which is something I fear IBM will not allow to happen. My biggest personal dislike in this area is IBM's use of PC-DOS and not the standard MS-DOS operating system. Had they standardized there would be no need for near and true compatible clones.

I guess what I would prefer people to say about the IBM until they become officially approved by the IEEE, is that their product has become a de facto standard by their users. Now the users do have considerable importances for considering standards, and by all means the number of machines out there may be as many as most other official standards, but until it is an official standard, it can not correctly be called one. It is also important, very important in fact, that the need for standards not detract from your ability to use the product. Non-standard products can be just as useful when sufficient information and support is available to allow you to do any task you desire. It is here that IBM and their clones really start to shine and where I start to support them. The physical product is really of little importance anymore, it is the software and the task it allows you to perform which is really important. It is this point which many people have missed, that they are in fact not a standard product, but one in which so many products exist that all other standards get ignored.

Some new products out which bear looking into (and writing articles about) and should help some users are next on my list of topics. Several new S-100 boards for emulating the IBM graphics and non-standard BIOS are now out on the market. For MAC users lots of expansions and cheaper versions are about to be released. IBM should have their new PC II out soon (still a few thousand warehoused PCs to sell first). All in all the next few months will see many new machines, most using 256K chips with 256K to one megabyte of memory. Now that the mask problems in the 80286 are almost all corrected, expect to see many more products there too. The 68000 is starting to gather more popularity, thanks to the MAC, but software is still a bit slow. For real speed freaks there are some FORTH machines coming out that will be the fastest micros yet. Expect to see 1200 baud modems below $200, IBM PC JR's for $150 and by the end of summer IBM PC's below $700 new.

For those people, like myself, without an unlimited pocketbook there are lots of cheap products on the market, one of which I need to spend some time with next. For as little as $39 you can get new Remex drives (double

(continued on page 31)